Model Checking Propositional Deontic Temporal Logic via a μ-calculus Characterization

Araceli Acosta[1], Cecilia Kilmurray[2],
Pablo F. Castro[2,3], and Nazareno M. Aguirre[2,3]

[1] Facultad de Matemática, Astronomía y Física, Universidad Nacional de Córdoba, Córdoba, Argentina, aacosta@famaf.unc.edu.ar
[2] Departamento de Computación, FCEFQyN, Universidad Nacional de Río Cuarto, Río Cuarto, Argentina, {ckilmurray,pcastro,naguirre}@dc.exa.unrc.edu.ar
[3] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)

Abstract. In this paper, we present a characterization of a propositional deontic temporal logic into μ-calculus. This logic has been proposed to specify and reason about fault tolerant systems, and even though is known to be decidable, no tool realizing its corresponding decision procedure has been developed. A main motivation for our work is enabling for the use of model checking, for analyzing specifications in this deontic temporal logic.
We present the technical details involved in the characterization, and prove that the model checking problem on the deontic temporal logic is correctly reduced to μ-calculus model checking. We also show that counterexamples are preserved, which is crucial for our model checking purposes. Finally, we illustrate our approach via a case study, including the verification of some properties using a μ-calculus model checker.

1 Introduction

With the increasing demand for highly dependable and constantly available systems, being able to reason about computer systems behavior in order to provide strong guarantees for software correctness, has gained considerable attention, especially for safety critical systems. In this context, a problem that deserves attention is that of capturing *faults*, understood as unexpected events that affect a system, as well as expressing and reasoning about the properties of systems in the presence of such faults.

Various researchers have been concerned with formally expressing fault tolerant behavior, and some formalisms and tools associated with this problem have been proposed [1, 20, 18, 7, 9, 12, 16, 15, 14, 13]. A particular trend in formal methods for fault tolerance, that concerns the work in this paper, is based on the observation that normal vs. abnormal behaviors can be treated as behaviors "obeying" and "violating" the rules of correct system conduct, respectively. From a logical point of view, this calls for a *deontic* approach, since deontic operators are especially well suited to express permission, obligation and prohibition, and thus to describe fault tolerant systems and their properties [6].

This idea has been exploited by various researchers in different ways (see for instance [6, 17, 8, 5]). In this paper, we are concerned with the approach taken in [5], where a propositional deontic logic (PDL) is introduced, and then extended with temporal logic features to express temporal behavior with a distinction between normative (i.e., non faulty) and non normative (i.e., faulty) behaviors, with straightforward applications to fault tolerance.

In the context of formal approaches to software development, it is generally recognized that powerful (semi-)automated analysis techniques are essential for a method to be effectively used in practice. In particular, the possibility of algorithmically checking whether a PDL formula, or a formula in its temporal extension DTL, holds for a given system is of great relevance for the take up of these logics as part of a formal method for fault tolerance. Fortunately, both PDL and its temporal extension DTL are known to be decidable [5]: a decision procedure for the logic DTL, based on a tableaux calculus, is proposed in [4]. However, the proposed decision procedure had a theoretical motivation, namely, proving that the logic was decidable; in fact, this tableaux calculus proved useful for investigating decidability and the logic's complexity, but was not devised as part of a tool for formal verification. Because of this fact, no practical considerations were taken in the definition of this decision procedure, and it has not been implemented in a tool for the analysis of fault tolerant specifications.

In this paper, we are concerned with the definition of a decision procedure for PDL and its extension DTL, with the purpose of being used for automated verification. Our approach consists of characterizing PDL/DTL in μ-calculus, and then use a μ-calculus model checker in order to verify whether a given system satisfies a fault tolerance property expressed in PDL/DTL. We thoroughly present our characterization of PDL/DTL in μ-calculus, and show how a fault tolerant system, captured by a deontic structure, can be analyzed for the satisfaction of PDL/DTL formulas, describing fault tolerant properties of the system. Moreover, we show that our translation from PDL/DTL into μ-calculus is correct, in the sense that the model checking problem in PDL/DTL is soundly reduced to model checking in μ-calculus. Moreover, we also show that counterexamples are maintained, meaning that every μ-calculus counterexample, resulting from the verification of a translated property on a translated model, can be mechanically traced back to a counterexample of the original deontic temporal specification. Finally, we provide some experimental results using the Mucke μ-calculus model checker [2], on a small case study illustrating how deontic structures capture systems with faults, and also illustrating our approach, as well as the details of our translation.

The paper proceeds as follows. In section 2 we present some preliminaries, including the syntax and semantics of PDL, as well as those of the μ-calculus. Section 3 introduces our translation from the core logic PDL to μ-calculus, and a proof of the correctness of the translation. Section 4 introduces DTL, consisting of PDL extended with CTL temporal operators, and Section 5 deals with the translation from DTL to μ-calculus, including a proof of the correctness of this characterization. The fact that counterexamples are preserved is also studied in

this section. Section 6 presents an example, consisting of a simple system with faults, and various sample properties regarding this faulty system and its fault tolerance mechanism. Finally, in Section 7 we draw some conclusions and discuss our current lines of work.

2 Preliminaries

2.1 A Propositional Deontic Logic (PDL)

We start this section with an introduction to the logic presented in [5], with some remarks. This logic is a propositional deontic action logic with boolean operators over actions, which comprises *vocabularies* and *actions*.

Definition 1 (Language). *A language or vocabulary is a tuple $\langle \Phi, \Delta \rangle$, where Φ is a finite set of propositional variables and Δ is a finite set of primitive actions.*

Primitive actions are used for describing the events that may occur during the execution of the system. Intuitively, events are identified with state changes. Primitive actions can be composed using the action operators $\emptyset$ (the abort action), $\mathbf{U}$ (the execution of any action of the system), $\sqcup$ (nondeterministic choice of two actions) and $\sqcap$ (parallel execution of two actions). Also, given an action α, $\neg\alpha$ denotes the execution of an alternative action to α (complementation). Given a set Δ_0 of primitive actions, the set Δ of action terms is defined as the closure of Δ_0 using the above action operators. From now on, Greek letters are used as action variables, and lowercase Roman letters are used as propositional variables.

Given a language $\langle \Phi_0, \Delta_0 \rangle$[4], the set Φ of *formulas* over this language is defined as the minimal set satisfying the following:

- $\Phi_0 \subseteq \Phi$,
- $\top, \bot \in \Phi$,
- if $\alpha, \beta \in \Delta$, then $\alpha =_{act} \beta \in \Phi$,
- if $\varphi, \psi \in \Phi$, then $\varphi \wedge \psi \in \Phi$ and $\neg\varphi \in \Phi$,
- if $\varphi \in \Phi$ and $\alpha \in \Delta$, then $\langle\alpha\rangle\varphi \in \Phi$,
- if $\alpha \in \Delta$, then $P(\alpha) \in \Phi$, $P_w(\alpha) \in \Phi$.

The models of PDL are given by deontic structures, which essentially consist of standard Kripke structures where each arc is colored with one of two colors: green, intuitively corresponding to allowed transitions, or red, intuitively denoting forbidden transitions (representing faults). Formally, given a language $\langle \Phi_0, \Delta_0 \rangle$, a deontic structure M over it is a tuple $\langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P} \rangle$, where:

- $\mathcal{W}$ is a set of states,
- $\mathcal{R} : \mathcal{E} \to \mathcal{W} \to \mathcal{W}$ is a function that for each $e \in \mathcal{E}$ returns a function $\mathcal{R}(e) : \mathcal{W} \to \mathcal{W}$. We say that $w \xrightarrow{e} w'$ when $(\mathcal{R}(e))(w) = w'$.

[4] We will use the 0 subscript when referring to languages for deontic formulas (in PDL or DTL).

– $\mathcal{E}$ is a non-empty set of events.
– $\mathcal{I}$ is an interpretation function, such that:
  • for each $p \in \Phi_0 : \mathcal{I}(p) \subseteq \mathcal{W}$,
  • for each $\alpha \in \Delta_0 : \mathcal{I}(\alpha) \subseteq \mathcal{E}$;

  function $\mathcal{I}$ must also satisfy the following:
  I.1 for each $\alpha_i \in \Delta_0 :| \mathcal{I}(\alpha_i) - \bigcup\{\mathcal{I}(\alpha_j) \mid \alpha_j \in (\Delta_0 - \alpha_i)\} | \leq 1$;
  I.2 for each $e \in \mathcal{E}$: if $e \in \mathcal{I}(\alpha_i) \cap \mathcal{I}(\alpha_j)$ where $\alpha_i \neq \alpha_j \in \Delta_0$, then

$$\bigcap\{\mathcal{I}(\alpha_k) \mid \alpha_k \in \Delta_0 \wedge e \in \mathcal{I}(\alpha_k)\} = \{e\};$$

  I.3 $\mathcal{E} = \bigcup_{\alpha_i \in \Delta_0} \mathcal{I}(\alpha_i)$.
– $\mathcal{P} \subseteq \mathcal{W} \times \mathcal{E}$ is a relationship indicating, for every state, the events that are allowed in it.

Due to space restrictions, we are unable to provide a thorough explanation of the intuitions behind the conditions on $\mathcal{I}$. We refer the reader to [5] for a more detailed explanation. It is worth remarking that the conditions on $\mathcal{I}$ imply that there is a one-to-one mapping between events and subsets of actions; basically, we can identify every subset of actions as the event that the parallel execution of these actions produces.

The interpretation mapping $\mathcal{I}$ can be extended to action terms, as follows:

– $\mathcal{I}(\neg\varphi) = \mathcal{W} - \mathcal{I}(\varphi)$,
– $\mathcal{I}(\varphi \wedge \psi) = \mathcal{I}(\varphi) \cap \mathcal{I}(\psi)$,
– $\mathcal{I}(\alpha \sqcup \beta) = \mathcal{I}(\alpha) \cup \mathcal{I}(\beta)$,
– $\mathcal{I}(\alpha \sqcap \beta) = \mathcal{I}(\alpha) \cap \mathcal{I}(\beta)$,
– $\mathcal{I}(\neg\alpha) = \mathcal{E} - \mathcal{I}(\alpha)$,
– $\mathcal{I}(\emptyset) = \emptyset$,
– $\mathcal{I}(\mathbf{U}) = \mathcal{E}$.

Satisfaction of formulas in a deontic structure is defined, given a deontic structure $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P} \rangle$ and a state $w \in \mathcal{W}$, as follows:

– $w, M \models_{PDL} p \iff w \in \mathcal{I}(p)$ with $p \in \Phi_0$,
– $w, M \models_{PDL} \alpha =_{act} \beta \iff \mathcal{I}(\alpha) = \mathcal{I}(\beta)$,
– $w, M \models_{PDL} \neg\varphi \iff$ not $w, M \models_{PDL} \varphi$,
– $w, M \models_{PDL} \varphi_1 \wedge \varphi_2 \iff w, M \models_{PDL} \varphi_1$ and $w, M \models_{PDL} \varphi_2$,
– $w, M \models_{PDL} \langle\alpha\rangle\varphi \iff$ there exists some $w' \in \mathcal{W}$ and $e \in \mathcal{I}(\alpha)$ such that $w \xrightarrow{e} w'$ and $w', M \models_{PDL} \varphi$,
– $w, M \models_{PDL} P(\alpha) \iff$ for all $e \in \mathcal{I}(\alpha)$, we have $\mathcal{P}(w, e)$,
– $w, M \models_{PDL} P_w(\alpha) \iff$ there exists some $e \in \mathcal{I}(\alpha)$ such that $\mathcal{P}(w, e)$.

From this definition, it becomes apparent that the "color" of arcs given by a deontic structure is captured by the relation $\mathcal{P}$. We have two deontic operators for permission, the standard one and "weak" permission. Obligation is defined in terms of these two, as follows:

$$O(\alpha) = P(\alpha) \ \wedge \ \neg P_w(\neg\alpha).$$

2.2 The μ-calculus

The μ-calculus, as other logics with fixed point operators, is an expressive formalism useful for investigating the expressiveness and algorithmic complexity of temporal and modal logics. A detailed introduction to μ-calculus can be found in [19]. In this section, we briefly recall the basic definitions regarding this formalism, since we use it as a target framework for interpreting the logic PDL, and its extension DTL, introduced in Section 4.

Given a language $\langle \Phi_1, \Delta_1 \rangle$[5] and a set V of variables, the set Φ_μ of $\mu-$calculus formulas is defined as follows:

- $\Phi_1 \subseteq \Phi_\mu$
- $V \subseteq \Phi_\mu$
- if $\varphi, \varphi_1, \varphi_2 \in \Phi_\mu$, then $\varphi_1 \wedge \varphi_2 \in \Phi_\mu$ and $\neg\varphi \in \Phi_\mu$
- if $\varphi \in \Phi_\mu$ and $\alpha \in \Delta_1$, then $\langle\alpha\rangle\varphi \in \Phi_\mu$ and $[\alpha]\varphi \in \Phi_\mu$
- if $\varphi \in \Phi_\mu$, then $\mu R.\varphi \in \Phi_\mu$ and $\nu R.\varphi \in \Phi_\mu$.

It is required that bound variables appear under an even number of negations.

Models of μ-calculus formulas are Kripke structures. More precisely, given a language $\langle \Phi_1, \Delta_1 \rangle$, a model for it is a tuple $M_\mu = \langle S, T, L \rangle$, where:

- S is a set of states.
- L is a function $L : \Phi_1 \to \wp(S)$ assigning to each proposition the set of states where it is true.
- T is a function $T : \Delta_1 \to \wp(S \times S)$ which, given an action, returns a binary relation whose domain and codomain is S. We say that $s \xrightarrow{a} s'$ if $(s, s') \in T(a)$.

Satisfaction in μ-calculus is defined as follows. Given a model M_μ, a state $s \in S$ and a formula φ without free variables, $s, M \models_\mu \varphi$ holds if and only if $s \in [\![\varphi]\!]_{M_\mu}\rho$, where ρ is a variable assignment (a mapping assigning values to variables). The interpretation $[\![\varphi]\!]_{M_\mu}\rho$ is recursively defined in the following way:

- $[\![p]\!]_{M_\mu}\rho = L(p)$ for $p \in \Phi_1$,
- $[\![R]\!]_{M_\mu}\rho = \rho(R)$ for $R \in V$,
- $[\![\neg\varphi]\!]_{M_\mu}\rho = S - [\![\varphi]\!]_{M_\mu}\rho$,
- $[\![\varphi \wedge \psi]\!]_{M_\mu}\rho = [\![\psi]\!]_{M_\mu}\rho \cap [\![\varphi]\!]_{M_\mu}\rho$,
- $[\![\langle a\rangle\varphi]\!]_{M_\mu}\rho = \{s \in S \mid \exists t[s \xrightarrow{a} t \wedge t \in [\![\varphi]\!]_{M_\mu}\rho]\}$,
- $[\![[a]\varphi]\!]_{M_\mu}\rho = \{s \in S \mid \forall t[s \xrightarrow{a} t \wedge t \in [\![\varphi]\!]_{M_\mu}\rho]\}$,
- $[\![\mu R.\varphi]\!]_{M_\mu}\rho$ is the least fixed point of the function $\tau : \wp(S) \to \wp(S)$, defined as:
$$\tau(T) = [\![\varphi]\!]_{M_\mu}\rho[R \mapsto T]^6,$$
- $[\![\nu R.\varphi]\!]_{M_\mu}\rho$ is defined in the same way, but using the greatest fixed point,

We will use $[\![\varphi]\!]_M$ instead of $[\![\varphi]\!]_{M_\mu}$ when no confusion is possible.

[5] We will use the 1 subscript when referring to languages for μ-calculus formulas, to distinguish these from those for the deontic logics.

[6] $(\rho[R \mapsto T])$ is the assignment ρ "updated" with the mapping $R \mapsto T$, i.e., it maps all elements as ρ, except for R which is mapped to T.

3 A μ-calculus Characterization of PDL

In this section, we start with our characterization of deontic temporal logic in terms of μ-calculus, by first dealing with the deontic logic PDL. As we explained in section 1, the purpose of this characterization, which we materialize via a translation Tr, is to be able to use μ-calculus model checkers for the verification of fault tolerance properties of systems, specified in PDL and its temporal extension DTL.

In what concerns this section, we expect to reduce PDL model checking to μ-calculus model checking, via Tr; that is, whenever we obtain that

$$Tr^m(w, M) \not\models_\mu Tr(\varphi)$$

then we must have that

$$w, M \not\models_{DPL} \varphi$$

and vice versa. Thus, we need the translation from PDL to μ-calculus to satisfy the following:

$$w, M \models_{PDL} \varphi \Longleftrightarrow Tr^m(w, M) \models_\mu Tr(\varphi).$$

Theoretically, translations between logics satisfying this property are called *forward morphisms* [10]. As we will show later on, this property allows us to guarantee that the model checking problem is preserved by translation.

Let us start by formally defining our translation.

Definition 2. *Let $\langle \Phi_0, \Delta_0 \rangle$ be a language, and $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P} \rangle$ be a deontic structure over that language. The mapping $Gen : \mathcal{E} \to \wp(\Delta_0)$ is defined as:*

$$Gen(e) = \{\alpha \mid \alpha \in \Delta_0 \wedge e \in \mathcal{I}(\alpha)\}$$

Given an event e, $Gen(e)$ corresponds to the set of actions whose parallel execution yield event e.

Lemma 1. *Gen is injective.*
Proof: *Let $e, e' \in \mathcal{E}$ be events such that $Gen(e) = Gen(e') = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$. First, notice that because of I.3, $n \neq 0$.*

If $n = 1$, we have that $e, e' \in \mathcal{I}(\alpha_1)$, and $\forall \alpha \in \Delta_0 - \{\alpha_1\} : e \notin \mathcal{I}(\alpha) \wedge e' \notin \mathcal{I}(\alpha)$. Then, $\{e, e'\} \subseteq \mathcal{I}(\alpha_1) - \bigcup\{\mathcal{I}(\alpha_i) \mid \alpha_i \in (\Delta_0 - \alpha_1)$, and because of I.1[7], it must be the case that $e = e'$.

If, on the other hand, $n > 1$, then there exist $\alpha_i, \alpha_j \in \Delta_0$ such that $\alpha_i \neq \alpha_j$, $\{e, e'\} \subseteq \mathcal{I}(\alpha_i)$ and $\{e, e'\} \subseteq \mathcal{I}(\alpha_j)$. But because of I.2[8], it must be the case that $e = e'$.

Let us now define the translation of PDL models into corresponding μ-calculus structures. This is, in fact, the first part of translation Tr.

[7] For each $\alpha_i \in \Delta_0 :\mid \mathcal{I}(\alpha_i) - \bigcup\{\mathcal{I}(\alpha_j) \mid \alpha_j \in (\Delta_0 - \alpha_i)\} \mid \leq 1$.

[8] For each $e \in \mathcal{E}$: if $e \in \mathcal{I}(\alpha_i) \cap \mathcal{I}(\alpha_j)$ where $\alpha_i \neq \alpha_j \in \Delta_0$, then $\bigcap\{\mathcal{I}(\alpha_k) \mid \alpha_k \in \Delta_0 \wedge e \in \mathcal{I}(\alpha_k)\} = \{e\}$.

Definition 3. *Let $\langle \Phi_0, \Delta_0 \rangle$ be a language, and $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P} \rangle$ and $w \in \mathcal{W}$ be a deontic structure. The mapping Tr^m is defined as:*

$$Tr^m(w, M) = w, M_\mu,$$

where $M_\mu = \langle S, T, L \rangle$ is a model of the language $\langle \Phi_1, \Delta_1 \rangle$ such that:

- $\Delta_1 = \wp(\Delta_0)$,
- $\Phi_1 = \Phi_0 \cup \{P_a \mid a \in \Delta_1\} \cup \{E_a \mid a \in \Delta_1\}$,
- $S = \mathcal{W}$,
- $T = \{w \overset{Gen(e)}{\longrightarrow} w' \mid w \overset{e}{\longrightarrow} w' \in \mathcal{R}\}$,
- $L(p) = I(p)$, *for every* $p \in \Phi_0$,
- $L(P_a) = \{w \mid \exists e \in \mathcal{E} \ : \ (w, e) \in \mathcal{P} \wedge Gen(e) = a\}$, *for every* $a \in \Delta_1$,
- $L(E_a) = \{s \mid \exists s' \ : \ s \overset{a}{\longrightarrow} s' \in T\}$.

It is worth noting that, in the above model translation, and since each event is the result of the parallel execution of a set of actions, we capture each event as the set of actions whose parallel execution produces it.

Now let us start dealing with the translation of formulas. First, notice that PDL formulas use action letters from Δ_0, whereas μ-calculus formulas use names coming from Δ_1 (i.e., subsets of Δ_0). So, our translation must relate both sets. In order to do so, we define the mapping *Set*, as follows.

Definition 4. *The mapping $Set : \Delta_0 \to \wp(\Delta_1)$ is defined as*

$$Set(\alpha) = \{a \mid a \in \Delta_1 \wedge \alpha \in a\}.$$

This mapping is extended recursively to action terms, in the following way:

- $Set(\emptyset) = \emptyset$,
- $Set(\boldsymbol{U}) = \Delta_1$,
- $Set(\neg\alpha) = \Delta_1 - Set(\alpha)$,
- $Set(\alpha \sqcup \beta) = Set(\alpha) \cup Set(\beta)$,
- $Set(\alpha \sqcap \beta) = Set(\alpha) \cap Set(\beta)$.

Finally, we are ready to define function Tr, that translates PDL formulas to μ-calculus formulas.

Definition 5. *The translation Tr, mapping PDL formulas to μ-calculus formulas, is defined as follows:*

- $Tr(p) = p$, *for every* $p \in \Phi_0$,
- $Tr(\top) = \top$,
- $Tr(\bot) = \bot$,
- $Tr(\neg\varphi) = \neg Tr(\varphi)$,
- $Tr(\varphi_1 \wedge \varphi_2) = Tr(\varphi_1) \wedge Tr(\varphi_2)$,
- $Tr(\alpha =_{act} \beta) = \bigwedge_{a \in (Set(\alpha) \cup Set(\beta)) - (Set(\alpha) \cap Set(\beta))} \neg E_a$,
- $Tr(\langle\alpha\rangle\varphi) = \bigvee_{a \in Set(\alpha)} \langle a \rangle Tr(\varphi)$,
- $Tr([\alpha]\varphi) = \bigwedge_{a \in Set(\alpha)} (E_a \to [a] Tr(\varphi))$,
- $Tr(P(\alpha)) = \bigwedge_{a \in Set(\alpha)} (E_a \to P_a)$,
- $Tr(P_w(\alpha)) = \bigvee_{a \in Set(\alpha)} P_a$.

3.1 On the Correctness of Tr

Let us briefly discuss some characteristics of the defined translation. Translations between logical systems have been extensively studied by the community of Institutions [11, 10]. In this context, logical systems are captured in abstract terms. The most usual kinds of translations between logical systems are the so called *morphisms* and *comorphisms* (or representations). In both of these cases, translations of models and formulas go in opposite directions. More precisely, a morphism between logical systems L and L' translates models of L into models of L', and formulas of L' into formulas of L, in a property-preserving way. Comorphisms, on the other hand, behave in the opposite way. Both cases then have the characteristics of Galois connections.

Our translation differs from morphisms and comorphisms, in the sense that it maps models and formulas "in the same direction". This kind of translation is called *forward morphism* [10]. Fortunately, this is the kind of morphism that we need, since forward morphisms are well suited for model checking reduction (the purpose of our translation). In section 5, we show that traces of a translated model can be traced back to the traces of the original model. Intuitively, this means that our translation preserves counterexamples, a crucial property for our model checking purposes.

The following theorem establishes that our translation is sound with respect to model checking reduction. It is proved straightforwardly by induction on the structure of PDL formulas, and resorting to their semantics and the definition of translation Tr. Due to space restrictions, the proof is not reproduced here.

Theorem 1. *Given a language $\langle \Phi_0, \Delta_0 \rangle$, a structure $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P} \rangle$ and a state $w \in \mathcal{W}$, we have:*

$$w, M \models \phi \Leftrightarrow Tr^m(w, M) \models_\mu Tr(\phi).$$

4 A Temporal Extension of PDL

The propositional deontic logic PDL that we introduced previously involves deontic operators for permission and obligation. In order to be able to express fault tolerance system properties, these deontic operators are combined with temporal ones, so that we can predicate about system executions. The temporal component of the resulting logic, that we call DTL, is a CTL-like logic. Besides the traditional CTL operators, this logic features an operator called *Done*, which enables one to talk about the immediate past. Intuitively, $Done(\alpha)$ is true when α was the last action executed in the system. Let us formally define this logic.

Definition 6. *Given a PDL language (Φ_0, Δ_0), the set of temporal formulas over it is defined as the minimal set Φ_T satisfying the following:*

- $\Phi \subseteq \Phi_T$,
- *if* $\alpha \in \Delta$, *then* $Done(\alpha) \in \Phi_T$,
- *if* $\varphi, \psi \in \Phi_T$, *then* $\varphi \wedge \psi \in \Phi_T$ *and* $\neg\varphi \in \Phi_T$,

– *if* $\varphi, \psi \in \Phi_T$, *then* $\mathtt{AG}\varphi \in \Phi_T, \mathtt{AN}\varphi \in \Phi_T, \mathtt{A}(\varphi \mathcal{U} \psi) \in \Phi_T, \mathtt{E}(\varphi \mathcal{U} \psi) \in \Phi_T$.

Note that Φ *represents the set of PDL formulas and* Δ *the set of actions terms as defined on page 3.*

Other CTL operators can be defined from the basic ones in the above definition, in the usual way. The temporal operators enable us to reason about execution traces. Let us define these traces formally.

Definition 7. *Given a structure* $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P} \rangle$ *and an initial state* w_0*, a trace or path is a labeled sequence* $s_0 \stackrel{e_0}{\rightarrow} s_1 \stackrel{e_1}{\rightarrow} s_2 \stackrel{e_2}{\rightarrow} \ldots$ *of states and events, such that for every* i*:* $s_i \stackrel{e_i}{\rightarrow} s_{i+1} \in \mathcal{R}$ *and* $s_0 = w$*.*

The set of all the traces starting in w *is denoted by* $\Sigma(w_0)$*.*

Given a trace π, we use the following notation to refer to states and events in a trace, to refer to subtraces, and to state that a trace is a prefix of another one:

– $\pi.i = s_i$,
– $\pi^{\rightarrow}.i = e_i$,
– $\pi[i, j]$ (where $i \leq j$) is the subpath $s_i \stackrel{e_i}{\rightarrow} \cdots \stackrel{e_{j-1}}{\rightarrow} s_j$,
– we say that $\pi' \preceq \pi$, if π' is an initial subpath of π; i.e. $s_0 \stackrel{e'_0}{\rightarrow} s'_1 \stackrel{e'_1}{\rightarrow} s'_2 \stackrel{e'_2}{\rightarrow} \ldots s'_k \preceq s_0 \stackrel{e_0}{\rightarrow} s_1 \stackrel{e_1}{\rightarrow} s_2 \stackrel{e_2}{\rightarrow} \ldots$ iff $s'_i = s_i$ for all $i \leq k$ and $e'_i = e_i$ for all $i < k$.

Let us define satisfaction for our deontic temporal logic (DTL). This definition extends the definition of satisfaction for PDL.

Definition 8. *Given a structure* $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P} \rangle$*, an initial state* $w_0 \in \mathcal{W}$ *and a path* $\pi \in \Sigma(w_0)$*, the relation* $\models_{DTL}$ *is defined as follows:*

– $\pi, i, M \models_{DTL} \varphi \iff \pi.i, M \models_{PDL}$, *if* $\varphi \in \Phi$,
– $\pi, i, M \models_{DTL} \neg\varphi \iff$ *not* $\pi, i, M \models_{DTL} \varphi$,
– $\pi, i, M \models_{DTL} \varphi \wedge \psi \iff \pi, i, M \models_{DTL} \varphi$ *and* $\pi, i, M \models_{DTL} \psi$,
– $\pi, i, M \models_{DTL} Done(\alpha) \iff i > 0$ *and* $\pi^{\rightarrow}.(i-1) \in \mathcal{I}(\alpha)$,
– $\pi, i, M \models_{DTL} AN\varphi \iff$ *for every* $\pi' \in \Sigma(\pi.0)$ *such that* $\pi[0, i] \preceq \pi'$ *we have that* $\pi', i+1, M \models_{DTL} \varphi$,
– $\pi, i, M \models_{DTL} AG\varphi \iff$ *for every* $\pi' \in \Sigma(\pi.0)$ *such that* $\pi[0, i] \preceq \pi'$ *we have that* $\forall j \geq i : \pi', j, M \models_{DTL} \varphi$,
– $\pi, i, M \models_{DTL} A(\varphi U \psi) \iff$ *for every* $\pi' \in \Sigma(\pi.0)$ *such that* $\pi[0, i] \preceq \pi'$ *we have that* $\exists j \geq i : \pi', j, M \models_{DTL} \psi$ *and* $\forall k : i \leq k < j : \pi', k, M \models_{DTL} \varphi$,
– $\pi, i, M \models_{DTL} E(\varphi U \psi) \iff$ *there exists* $\pi' \in \Sigma(\pi.0)$ *such that* $\pi[0, i] \preceq \pi'$ *we have that* $\exists j \geq i : \pi', j, M \models_{DTL} \psi$ *and* $\forall k : i \leq k < j : \pi', k, M \models_{DTL} \varphi$.

Given a structure $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P} \rangle$, an initial state $w_0 \in \mathcal{W}$ and a formula φ, we say that

$$M, w_0 \models \varphi$$

if and only if

$$\forall \pi \in \Sigma(w_0) : \pi, 0, M \models_{DTL} \varphi.$$

5 Translating DTL Formulas to μ-calculus

Now that we have extended PDL with temporal operators, obtaining the logic DTL, we need also to extend the definition of Tr, to cope with temporal formulas. Let us first deal with the translation of models, via a translation that we refer to as Tr^{tm}. This involves explicitly identifying the initial states, which we achieve via a function called Tr^s in the definition below.

Definition 9. *Let (Φ_0, Δ_0) be a language, $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P} \rangle$ a structure over that language, and $w_0 \in \mathcal{W}$ an initial state in M. The functions Tr^{tm} and Tr^s are defined as follows:*

$$Tr^{tm}(M) = M_\mu$$
$$Tr^s(w) = s^w_\emptyset$$

where $M_\mu = \langle S, T, L \rangle$, a μ-calculus model for the language $\langle \Phi_1, \Delta_1 \rangle$, and $s^w_\emptyset \in S$ are obtained in the following way:

- $\Delta_1 = \wp(\Delta_0)$,
- $\Phi_1 = \Phi_0 \cup \{P_a \mid a \in \Delta_1\} \cup \{E_a \mid a \in \Delta_1\} \cup \{D_a \mid a \in \Delta_1\}$,
- $S = \{s^w_a \mid w \in \mathcal{W} \wedge a \in (Im(Gen) \cup \{\emptyset\})\}$, [9]
- $T = \{s^w_a \xrightarrow{Gen(e)} s^{w'}_{Gen(e)} \mid w \xrightarrow{e} w' \in \mathcal{R} \wedge a \in (Im(Gen) \cup \{\emptyset\})\}$,
- $L(p) = \mathcal{I}(p)$ *for every* $p \in \Phi_0$,
- $L(P_a) = \{w \mid \exists e \in \mathcal{E} \ : \ (w, e) \in \mathcal{P} \wedge Gen(e) = a\}$ *for every* $a \in \Delta_1$,
- $L(E_a) = \{s \mid \exists s' \ : \ s \xrightarrow{a} s' \in T\}$,
- $L(D_a) = \{s^w_a \mid w \in \mathcal{W}\}$.

Let us now deal with the translation of DTL formulas into μ-calculus. Because of the operator *Done*, this translation requires characterizing the last executed action, as it can be seen in the next definition.

Definition 10. *The translation Tr from PDL to μ-calculus is extended to DTL formulas in the following way:*

- *for $\varphi \in \Phi$, $Tr(\varphi)$ is defined as described in Definition 5,*
- $Tr(\neg\varphi) = \neg Tr(\varphi)$,
- $Tr(\varphi \wedge \psi) = Tr(\varphi) \wedge Tr(\psi)$,
- $Tr(Done(\alpha)) = \bigvee_{a \in Set(\alpha)} D_a$,
- $Tr(AN\varphi) = Tr([U]\varphi)$,
- $Tr(EN\varphi) = Tr(\langle U \rangle \varphi)$,
- $Tr(AG\varphi) = \nu R.(Tr(\varphi) \wedge \bigwedge_{a \in \Delta_1} [a]R)$,
- $A(\varphi U \psi) = \mu R.(Tr(\psi) \vee (Tr(\varphi) \wedge \bigwedge_{a \in \Delta_1} [a]R))$,
- $E(\varphi U \psi) = \mu R.(Tr(\psi) \vee (Tr(\varphi) \wedge \bigvee_{a \in \Delta_1} \langle a \rangle R))$.

[9] $Im(f)$ denotes the image of f.

5.1 On the Correctness of the Extended Tr

The correctness of the translation extended to DTL is proved by generalizing the theorem regarding the correctness of the translation on PDL. Let us first define the translation of paths.

Definition 11. *The mapping Tr^p is defined as follows:*

$$Tr^p(w_0 \stackrel{e_0}{\to} w_1 \stackrel{e_1}{\to} w_2 \stackrel{e_2}{\to} \dots) = s_{\emptyset}^{w_0} \stackrel{Gen(e_0)}{\to} s_{Gen(e_0)}^{w_1} \stackrel{Gen(e_1)}{\to} s_{Gen(e_1)}^{w_2} \stackrel{Gen(e_2)}{\to} \dots.$$

Notice that, since Gen is injective, given a translated trace π in the target model, we have a unique trace π' in the original model, such that $Tr^p(\pi') = \pi$. In other words, the translation of traces is invertible. On the other hand, Tr^p is surjective by construction, and therefore we obtain the following Lemma.

Lemma 2. *Tr^p is a bijection.*

The following theorem enables us to relate validities in μ-calculus with validities in the deontic-temporal logic DTL.

Theorem 2. *Given a language $\langle \Phi_0, \Delta_0 \rangle$, a structure $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P} \rangle$, an initial state $w_0 \in \mathcal{W}$, and a formula ϕ, $s_a^w \in [\![Tr(\phi)]\!]_{Tr^{tm}(M)}$ if and only if $\forall \pi, i \cdot \pi, i, M \models_{DTL} \phi$ when $s_a^w = Tr^p(\pi).i$.*

This theorem implies the correctness of the translation of DTL formulas, in a straightforward way.

Corollary 1. *Given a language $\langle \Phi_0, \Delta_0 \rangle$, a structure $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P} \rangle$, an initial state $w_0 \in \mathcal{W}$ and a formula φ, the following holds:*

$$w_0, M \models_{DTL} \varphi \leftrightarrow s_{\emptyset}^{w_0}, Tr^{tm}(M) \models_{\mu} Tr(\varphi)$$

It is worthwhile remarking that, since there is a bijection between paths (and states and translated states maintain equivalent properties), for every path in a translated model that is a counterexample of a given translated property, a trace in the original model can be systematically constructed, which is guaranteed to be a counterexample of the original property. In other words, counterexamples that are obtained using μ-calculus model checkers can be systematically translated into counterexamples of the original DTL specification.

6 An Example

In this section, we describe a small example illustrating our translation. Moreover, we will use the Mucke model checker [2] in order to verify fault tolerance properties over this example. Our example consists of a simple communication scenario, composed of a producer, a consumer, and a channel used for communicating these two. The structures in Figure 1 graphically depict these components. In order to incorporate faults, and as a consequence the need for fault tolerance,

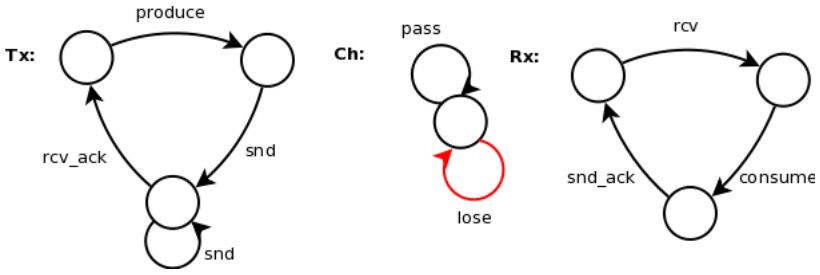

Fig. 1. A producer, a consumer, and a faulty channel.

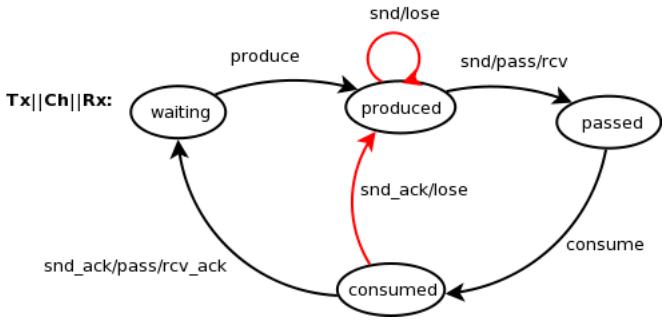

Fig. 2. Composition of the producer, consumer and faulty channel.

the channel is assumed to have "noise", and therefore some messages might be lost. In order to cope with this fault, the producer and consumer communicate using a typical protocol that forces the sender to resend the last message until an acknowledgement is received. The only forbidden action is the one corresponding to "losing" a message held in the channel. The deontic operators will allow us to indirectly refer to executions exercising normal (permitted) and faulty (forbidden) transitions.

In this paper, we do not deal in detail with the way components are synchronized. But basically, if an action, normal or faulty, is synchronized with a faulty action, the composite action is also considered faulty. Therefore, and since the use of the channel (sending a message) can be synchronized with the correct "passing" of the message and with the unfortunate event of "losing" it, the latter will be considered faulty. For this reason, in the system resulting of the composition of the producer, the consumer, and the channel (see Figure 2), the faulty actions are `snd/lose` and `snd_ack/lose`.

In this example, the language of the deontic structure is given by the following sets of propositional variables and actions:

$\Phi_0 = \{$`waiting, produced, passed, consumed`$\}$
$\Delta_0 = \{$`produce, consume, snd, rcv, snd_ack, rcv_ack, pass, lose`$\}$

The deontic structure for the example is formally the following:

– the set of states $\mathcal{W}$ and the transition system R are defined as in Figure 2,
– the set of events is the following:

$$\mathcal{E} = \{\texttt{produce, consume, snd/lose, snd/pass/rcv, snd\_ack/lose, snd\_ack/pass/rcv\_ack}\},$$

– the interpretation of the propositional variables is described in Figure 2, and the interpretation of actions is given by
  • $\mathcal{I}(\texttt{produce}) = \{\texttt{produce}\}$,
  • $\mathcal{I}(\texttt{consume}) = \{\texttt{consume}\}$,
  • $\mathcal{I}(\texttt{snd}) = \{\texttt{snd/lose, snd/pass/rcv}\}$,
  • $\mathcal{I}(\texttt{rcv}) = \{\texttt{snd/pass/rcv}\}$,
  • $\mathcal{I}(\texttt{lose}) = \{\texttt{snd/lose, snd\_ack/lose}\}$,
  • $\mathcal{I}(\texttt{pass}) = \{\texttt{ snd/pass/rcv, snd\_ack/pass/rcv\_ack}\}$,
  • $\mathcal{I}(\texttt{rcv\_ack}) = \{\texttt{snd\_ack/pass/rcv\_ack}\}$,
  • $\mathcal{I}(\texttt{snd\_ack}) = \{\texttt{snd\_ack/lose, snd\_ack/pass/rcv\_ack}\}$.
– Allowed events are all the arrows in Figure 2, except for those labeled with `snd/lose` and `snd_ack/lose`.

Now let us describe the resulting μ-calculus model, obtained by translating the above deontic structure. Following our description of the translation, the resulting model is composed of the following ingredients:

– $\Delta_1 = \{\texttt{produce, consume, snd/lose, snd/pass/rcv, snd\_ack/lose, snd\_ack/pass/rcv\_ack }\}$,
– $\Phi_1 = \Phi_0 \cup \{P_a \mid a \in \Delta_1\}$,
– $S = \mathcal{W} = \{waiting, produced, passed, consumed\}$,
– T is as described in Figure 2,
– $L(p)$ is as described in Figure 2,
– $L(P_a)$ is true in those states in which we have a transition labeled with a, other of the two faulty ones,
– $L(E_a)$ is true in those states in which we have a transition labeled with a.

6.1 Expressing Properties of Producer-Consumer

Now that the model has been fully described, we may start specifying intended properties of this model. Some interesting sample properties are the following:

– Whenever the system is in the waiting state, it is obliged to produce an item.
– After the production of an item, and if no fault occurs, then the system is obliged to consume.
– If an item has been produced, then no new item can be produced until the current item has been consumed.
– When a item has been consumed, then no additional items are consumed until some new item is produced.
– When an item is produced, all possible ways of performing the send action are allowed.

These properties are captured using DTL rather straightforwardly, thanks to the deontic component of this logic:

P1 $waiting \rightarrow O(produce)$
P2 $[\ produce\ ]\ [\ \neg lose\ ]\ O(consume)$
P3 $produced \rightarrow \mathbf{A(}\ O(\neg produce)\ \mathbf{U}\ consumed\ \mathbf{)}$
P4 $consumed \rightarrow \mathbf{ANA(}\ \neg consumed\ \mathbf{W}\ produced\ \mathbf{)}$
P5 $[\ produce\ ]\ P(snd)$

In order to model check these properties for the above described model, we have to use our translation Tr from DTL to μ-calculus. This translation gives us the following μ-calculus formulas for the above DTL properties:

TP1 $waiting \rightarrow P_{\texttt{produce}} \wedge \bigwedge_{a\in\delta} \neg P_a$,where:

$\delta = \{$`consume, snd/lose, snd/pass/rcv, snd_ack/lose, snd_ack/pass/rcv_ack` $\}$

TP2 $[\ \texttt{produce}\ ]\ \left(\bigwedge_{a\in\delta} [\ a\ ]\ \left(P_{\texttt{consume}} \wedge \bigwedge_{a\in\delta'} \neg P_a\right)\right)$, where:

$\delta = \{$`produce, consume, snd/pass/rcv, snd_ack/pass/rcv_ack` $\}$
$\delta' = \{$`produce, snd/lose, snd/pass/rcv, snd_ack/lose, snd_ack/pass/rcv_ack` $\}$.

TP3 $produced \rightarrow \mu R.\left(consumed \vee \left(\bigwedge_{a\in\delta}(E_a \rightarrow P_a) \wedge \neg P_{\texttt{produce}} \wedge \bigwedge_{a\in\Delta_1} [\ a\ ]\ R\right)\right)$, where

$\delta = \{$`consume, snd/lose, snd/pass/rcv, snd_ack/lose, snd_ack/pass/rcv_ack` $\}$

TP4 $consumed \rightarrow \bigwedge_{a\in\Delta_1} [\ a\ ]\ ((\mu R.(produced \vee (\neg consumed \wedge \bigwedge_{a\in\Delta_1} [\ a\ ]\ R)) \vee \nu R.(\neg consumed \wedge \bigwedge_{a\in\Delta_1} [\ a\ ]\ R))$
TP5 $[\ \texttt{produce}\ ]\ \bigwedge_{a\in\delta}(E_a \rightarrow P_a)$, where:

$\delta = \{$ `snd/lose, snd/pass/rcv, snd_ack/lose, snd_ack/pass/rcv_ack` $\}$

The careful reader might notice that, technically, formula **TP4** is not a valid μ-calculus formula, since an odd number of negations appear under the scope of fix point operators. However, our use of negation in this case is simply as a shorthand: $\neg consumed$ can be positively described as the disjunction of all states different from *consumed*.

We employed the Mucke model checker to verify these formulas. Properties **P1**, **P2**, **P4** were found to hold in the presented model, whereas **P3** and **P5** are invalid. The invalidity of **P3** could seem surprising at first sight; the counterexample found by the model checker is the following:

$$produced \overset{snd/lose}{\rightarrow} produced \overset{snd/pass/rcv}{\rightarrow} passed \overset{consume}{\rightarrow} consumed$$

Notice that the transition labelled by *snd/lose* is not allowed, and therefore it is not obligatory, falsifying $O(\neg produce)$.

7 Conclusions and Future Work

We have presented an approach for model checking a propositional temporal-deontic logic, with applications to fault tolerance verification, based on a characterization of this logic into the μ-calculus. This characterization is materialized via two translations, one capturing deontic structures as Kripke structures for μ-calculus, and the other translating formulas in the deontic-temporal logic into μ-calculus. This translation is shown to be correct, in the sense that the model checking problem in the deontic-temporal logic is reduced to model checking in μ-calculus. Moreover, we also show that counterexamples are also maintained, meaning that every μ-calculus counterexample, resulting from the verification of a translated property on a translated model, can be mechanically traced back to a counterexample of the original deontic temporal specification. Although the deontic temporal logic subject of study in this paper was known to be decidable, the decision procedure for it was not originally conceived for model checking purposes, and therefore none of the practical considerations we had in our approach were previously taken. In our opinion, this justifies the relevance of our work, aiming at automated verification of fault tolerance models.

We also provided a simple example illustrating various points. First, it illustrates the use of deontic structures for capturing systems with faults; second, it allows us to show how fault tolerance properties are straightforwardly captured by the combination of deontic and temporal operators; and third, it allowed us to illustrate the translation from deontic temporal logic into μ-calculus. Moreover, we employed the Mucke model checker in order to verify some sample properties on the presented model, and found a nontrivial counterexample on a property that was supposed to hold in the model.

As work in progress, we are developing a tool for fault tolerance system description and verification, which this work is part of. We are also studying the complexity of the model checking problem for PDL/DTL in relation to our translation. It is known that SAT for PDL is in PSPACE [3], but we do not have yet results regarding DTL, nor the complexity of these logics' model checking. Other concerns we are currently investigating are compositional reasoning on the presented temporal deontic logic, and integrating the presented model checking approach with the deductive mechanisms for verification presented in [3].

Acknowledgements

The authors would like to thank Pedro D'Argenio and the anonymous referees for their helpful comments. This work was partially supported 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. C. Bernardeschi, A. Fantechi, and S. Gnesi. Model checking fault tolerant systems. *Softw. Test., Verif. Reliab.*, 12(4):251–275, 2002.

2. A. Biere. μcke - efficient μ-calculus model checking. In O. Grumberg, editor, *CAV*, volume 1254 of *Lecture Notes in Computer Science*, pages 468–471. Springer, 1997.
3. P. F. Castro. *Deontic Action Logics for the Specification and Analysis of Fault-Tolerance*. PhD thesis, McMaster University, Department of Computing and Software, 2009.
4. P. F. Castro and T.S.E. Maibaum. A tableaux system for deontic action logic. In *Proceedings of 9th International Conference on Deontic Logic in Computer Science , Luxembourg*. Springer-Verlag, 2008.
5. P. F. Castro and T.S.E. Maibaum. Deontic action logic, atomic boolean algebra and fault-tolerance. *Journal of Applied Logic*, 7(4):441–466, 2009.
6. J. Coenen. *Formalisms for Program Reification and Fault Tolerance*. PhD thesis, Tenische Universiteit Eindhoven, 1994.
7. F. Cristian. A rigorous approach to fault-tolerant programming. *IEEE Trans. Software Eng.*, 11:23–31, 1985.
8. T. French, J. Christopher McCabe-Dansted, and M. Reynolds. A temporal logic of robustness. In B. Konev and F. Wolter, editors, *FroCos*, volume 4720 of *Lecture Notes in Computer Science*, pages 193–205. Springer, 2007.
9. F. Gärtner. Specification for fault-tolerance: A comedy of failures. Technical report, Darmstadt University of Technology, 1998.
10. J. A. Goguen and G. Rosu. Institution morphisms. *Formal Asp. Comput.*, 13(3-5):274–307, 2002.
11. J.A. Goguen and R.M. Burstall. Institutions: Abstract model theory for specification and programming. In *Journal of the Association of Computing Machinery*, 1992.
12. T. Janowski. On bisimulation, fault-monotonicity and provable fault-tolerance. In *AMAST*, pages 292–306, 1997.
13. L. Lamport and S. Merz. Specifying and verifying fault-tolerant systems. In *Formal Techniques in Real-Time and Fault-Tolerant Systems, Third International Symposium Organized Jointly with the Working Group Provably Correct Systems - ProCoS*, pages 41–76, 1994.
14. L. A. Laranjeira, M. Malek, and R. M. Jenevein. Nest: A nested-predicate scheme for fault tolerance. *IEEE Trans. Computers*, 42:1303–1324, 1993.
15. Z. Liu and M. Joseph. Specification and verification of fault-tolerance, timing, and scheduling. *ACM Trans. Program. Lang. Syst.*, 21(1):46–89, 1999.
16. A. Lomuscio and M. J. Sergot. A formalisation of violation, error recovery, and enforcement in the bit transmission problem. *Journal of Applied Logic*, 2:93–116, 2004.
17. J. Magee and T.S.E. Maibaum. Towards specification, modelling and analysis of fault tolerance in self managed systems. In *Proceeding of the 2006 international workshop on self-adaptation and self-managing systems*, 2006.
18. F. Schneider, S. M. Easterbrook, J. R. Callahan, and G. J. Holzmann. Validating requirements for fault tolerant systems using model checking. In *3rd International Conference on Requirements Engineering (ICRE '98)*, 1998.
19. K. Schneider. *Verification of Reactive Systems, Formal Methods and Algorithms*. Springer, 2004.
20. T. Yokogawa, T. Tsuchiya, and T. Kikuno. Automatic verification of fault tolerance using model checking. In *Pacific Rim International Symposium on Dependable Computing.*, 2001.