dCTL: A Branching Time Temporal Logic for Fault-Tolerant System Verification

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

[1] Departamento de Computación, FCEFQyN, Universidad Nacional de Río Cuarto, Río Cuarto, Córdoba, Argentina
{pcastro,ckilmurray,naguirre}@dc.exa.unrc.edu.ar

[2] Facultad de Matemática, Astronomía y Física, Universidad Nacional de Córdoba, Córdoba, Argentina
aacosta@famaf.unc.edu.ar

[3] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina

Abstract. With the increasing demand for highly dependable and constantly available systems, being able to reason about *faults* and their impact on systems is gaining considerable attention. In this paper, we are concerned with the provision of a logic especially tailored for describing fault tolerance properties, and supporting automated verification. This logic, which we refer to as dCTL, employs *temporal deontic operators* in order to distinguish "good" (normal) from "bad" (faulty) behaviors, using deontic permission, prohibition and obligation combined in a novel way with temporal operators. These formulas are interpreted over transition systems, in which normal executions are distinguished from faulty ones. Furthermore, we show that this logic is sufficiently expressive to describe various common properties of interest in fault tolerant systems, and show that it features some desirable characteristics that make it suitable for analysis. Indeed, even though we show that the logic is more expressive than CTL, we prove that it maintains the time complexity of the model checking problem for CTL. The logic, its expressiveness and its use to express properties of fault tolerant systems, are illustrated via some case studies.

Keywords: Formal Methods, Fault Tolerance, Temporal Logic, Model Checking.

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 these faults.

Various researchers have been concerned with formally expressing fault tolerant behavior, and some formalisms and tools associated with this problem have been proposed [6]. Some recent approaches include the use of model checking for analyzing fault tolerant systems [11], and the employment of synthesis mechanisms for systematically producing controllers that help to achieve fault tolerance [8]. A particular trend in formal methods for fault tolerance, that we take as a starting point 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. This leads to a straightforward application of *deontic* operators (operators to express permission, obligation and prohibition) for separating normal from abnormal behaviors, and thus for expressing fault tolerant systems and their properties [7]. This idea has been exploited by various researchers in different ways, e.g., for extending a Hoare logic with the possibility of expressing properties of programs in the presence of exceptions [7], for specifying normal behavior of components in distributed systems [13], for specifying fault tolerant systems and their intended properties [3], and the extension of temporal logics with obligation so that robustness can be expressed [10], amongst others.

The work we present in this paper is related to the above mentioned deontic logic approaches to fault tolerance specification and reasoning. We propose a logic especially tailored for describing fault tolerance properties based on the use of deontic operators, with an emphasis on expressing intended (temporal) properties of fault tolerant systems, rather than (axiomatically) prescribing component/system behavior. We then share the motivation of related works such as [13,3], but components will be described using behavioral models such as transition systems, and the logic will be reserved for expressing properties regarding these systems. We maintain a strong concern on automated verification of these properties. Indeed, this logic, which we refer to as dCTL, is composed of CTL and deontic operators for distinguishing "good" (normal) from "bad" (faulty) behaviors, as other deontic approaches, but the way in which temporal and deontic operators are combined makes the logic suitable for analysis. Our proposed dCTL logic is more expressive than CTL, which as we will argue makes it useful for describing common properties of interest in the context of fault tolerant systems, but it preserves the complexity of the model checking problem for CTL, as we show in this paper. Thus, it constitutes a good candidate for describing temporal properties of fault tolerant systems, when the intention is to use model checking for their analysis. This is so especially compared to related temporal-deontic approaches such as RoCTL* [10,14], for which model checking is currently reduced to CTL* model checking, and thus is significantly less efficient.

We provide a number of case studies which enable us to illustrate the use of the logic, and its expressive power. These case studies, though small, represent simple models of common situations in fault tolerance, and are useful for assessing the expressiveness of the logic. They are presented simply as transition systems in which normal states (those resulting from a normal transition), are distinguished from abnormal ones (those resulting from a fault).

2 Preliminaries

In this section, we reproduce some basic definitions and facts regarding Kripke structures and CTL, which are necessary in the presentation of our logic.

2.1 Kripke Structures

Kripke structures are a standard vehicle for interpreting modal or temporal logic formulas as well as for characterizing the operational behavior of reactive systems [6]. Let AP be a set of atomic propositions. A Kripke structure over AP is a 4-tuple $\langle S, I, R, L\rangle$, where S is a set of elements called *states*, $I \subseteq S$ is a set of *initial states*, $R \subseteq S \times S$ is a *transition* relation between states, and $L : S \to 2^{AP}$ is an interpretation function, which indicates the set of atomic propositions that hold in each state.

Given a Kripke structure $M = \langle S, I, R, L\rangle$, the interpretation of logical connectives and modal operators in a modal logic can typically be defined by resorting to L and the structure of R. For temporal logics, it is usually necessary to employ the notion of *trace* to define the semantics of some operators. A *trace* is simply a maximal sequence of states, adjacent with respect to R. When a trace starts in an initial state, it is called an *execution* of M, with *partial* executions corresponding to non-maximal sequence of adjacent states. 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 $\sigma[i..]$. Finally, we will denote by $\mathcal{U}_M$ the set of all traces, i.e., maximal sequences of adjacent states, of M.

Without loss of generality, it can be assumed that every state has a successor, as is customary in various temporal logics [2].

Colored Kripke Structures. We define a *colored Kripke structure* as a 5-tuple $\langle S, I, R, L, \mathcal{N}\rangle$, where $\langle S, I, R, L\rangle$ is a Kripke structure and $\mathcal{N} \subseteq S$ is a set of *normal* states. Arcs leading to abnormal states can be thought of as faulty transitions, our representation of *faults* (similar approaches to formally model faults can be found in the literature, e.g., [12]). Then, normal executions will be those transiting only through normal states. The set of normal executions will be denoted by $\mathcal{NT}$. In this paper, 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 normal. This guarantees that every system has at least one normal execution, i.e., that $\mathcal{NT} \neq \emptyset$.

2.2 Computation Tree Logic

Computation Tree Logic (CTL) is a branching time temporal logic with important applications in model checking [5]. This logic allows for the description of properties over Kripke structures, by complementing propositional connectives with path quantifiers and temporal operators, combined in a certain restricted way. It is a logic of "computation trees" since it allows one to express properties referring to the tree that is constructed from a Kripke structure, starting from an initial state, and unfolding the structure to form a (typically infinite) tree.

Let us describe the syntax of the logic. Let AP be a set $\{p_0, p_1, \ldots\}$ of atomic propositions; the set Φ of CTL well formed formulas is recursively defined as:

$$\Phi ::= \top \mid p_i \mid \neg\Phi \mid \Phi \rightarrow \Phi \mid \mathsf{EX}\Phi \mid \mathsf{AX}\Phi \mid \mathsf{E}(\Phi\,\mathcal{U}\,\Phi) \mid \mathsf{A}(\Phi\,\mathcal{U}\,\Phi)$$

CTL formulas are interpreted on states over a Kripke structure. Given a Kripke structure $M = \langle S, I, R, L\rangle$, and a state $s \in S$, the semantics of CTL formulas is defined as follows:

- $M, s \models \top$
- $M, s \models p_i \Leftrightarrow p_i \in L(s)$, where $p_i \in AP$.
- $M, s \models \neg\varphi \Leftrightarrow$ not $M, s \models \varphi$.
- $M, s \models \varphi \rightarrow \varphi' \Leftrightarrow (M, s \models \neg\varphi)$ or $(M, s \models \varphi')$.
- $M, s \models \mathsf{EX}\varphi \Leftrightarrow$ for some traces σ such that $\sigma[0] = s$, $M, \sigma[1] \models \varphi$.
- $M, s \models \mathsf{AX}\varphi \Leftrightarrow$ for all traces σ such that $\sigma[0] = s$, $M, \sigma[1] \models \varphi$.
- $M, s \models \mathsf{E}(\varphi\,\mathcal{U}\,\varphi') \Leftrightarrow$ for some traces σ such that $\sigma[0] = s$, there exists a $j \geq 0$ such that $M, \sigma[j] \models \varphi'$, and for every $0 \leq k < j$, $M, \sigma[k] \models \varphi$.
- $M, s \models \mathsf{A}(\varphi\,\mathcal{U}\,\varphi') \Leftrightarrow$ for all traces σ such that $\sigma[0] = s$, there exists a $j \geq 0$ such that $M, \sigma[j] \models \varphi'$, and for every $0 \leq k < j$ satisfies $M, \sigma[k] \models \varphi$.

Some model checkers, particularly SMV, employ CTL as a language for expressing temporal properties of systems. The model checking problem for this logic is known to be linear on the size of the system and the formula being verified, as opposed to the case of CTL*, a more expressive computation tree logic for which the model checking problem is exponential on the size of the verified formula [6].

3 A Deontic Computation Tree Logic: dCTL

In this section we introduce dCTL, the logic that will be employed in order to specify, and later on verify, properties of fault tolerant systems. Formulas in this logic refer to properties of behaviors of colored Kripke structures, as defined in the previous section, 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 CTL, with its novel part being the deontic operators $\mathbf{O}(\psi)$ (obligation), $\mathbf{P}(\psi)$ (permission), and $\mathbf{R}(\psi)$ (repair or recovery), which apply on a certain kind of path formula ψ. The intention of these operators is to capture the corresponding notion of *obligation*, *permission* and *repair* 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 any faults, starting from the current state and along which ψ holds.
- $\mathbf{R}(\psi)$: property ψ holds in every future faulty state, i.e., resulting from the immediate occurrence of a fault.

Clearly, obligation and permission will enable us to express intended properties which should hold in *all* normal behaviors and *some* normal behaviors, respectively. Repair, on the other hand, will enable us to express properties that should hold when faults occur; they will mainly serve the purpose of imposing restrictions on what should happen when faults occur, so that certain properties can be guaranteed.

These deontic operators have an implicit *temporal* character, since ψ is a path formula. As it will be made clearer later on, these operators, in combination with path formulas of the form $\psi \rightsquigarrow \psi'$ (operator $\rightsquigarrow$ is an *implication* between trace properties), provide some additional expressiveness with respect to CTL, without augmenting the expressiveness of the standard CTL operators A and E. As we will argue in the next section, these operators, used in a combined way, will be useful to state some fault tolerance properties straightforwardly.

Let us present the syntax of our logic. Let AP be a set $\{p_0, p_1, \ldots\}$ of atomic propositions; the sets Φ and Ψ of state and path formulas, respectively, are mutually recursively defined as follows:

$$\begin{aligned}\Phi ::= {} & \top \mid p_i \mid \neg\Phi \mid \Phi \rightarrow \Phi \mid \mathsf{A}(\Psi \rightsquigarrow \Psi) \mid \mathsf{E}(\Psi \rightsquigarrow \Psi) \mid \mathbf{O}(\Psi \rightsquigarrow \Psi) \mid \mathbf{P}(\Psi \rightsquigarrow \Psi) \\ & \mid \mathbf{R}(\Psi \rightsquigarrow \Psi')\end{aligned}$$

$$\Psi ::= \mathsf{X}\Phi \mid \Phi\, \mathcal{U}\, \Phi \mid \Phi\, \mathcal{W}\, \Phi$$

Other boolean connectives (here, state operators), such as $\wedge$, $\vee$, etc., can be 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. Notice however that both CTL quantifiers and deontic operators apply to formulas involving the operator $\rightsquigarrow$. This operator relates two path formulas, and it represents a conditional. For instance, $\mathbf{O}(\psi \rightsquigarrow \psi')$ indicates that, for every normal trace σ starting in the current state, if σ satisfies ψ then it also satisfies ψ'. From a more technical perspective, which will be made clearer in later sections, the operator $\rightsquigarrow$ enables us to restrict the way in which path formulas can be combined in the scope of a state operator (a mechanism also exploited in other logics, particularly CTL^2). This will be essential for extending the expressiveness of CTL while retaining its model checking complexity.

Let us formally state the semantics of our logic. We start by defining the relationship $\vDash$, formalizing the satisfaction of dCTL state formulas in colored Kripke structures:

- $M, s \vDash \top$.
- $M, s \vDash p_i \Leftrightarrow p_i \in L(s)$, where $p_i \in AP$.
- $M, s \vDash \neg\varphi \Leftrightarrow$ not $M, s \vDash \varphi$.
- $M, s \vDash \varphi \rightarrow \varphi' \Leftrightarrow (M, s \vDash \neg\varphi)$ or $(M, s \vDash \varphi')$.
- $M, s \vDash \mathsf{A}(\psi \rightsquigarrow \psi') \Leftrightarrow M, \sigma \vDash \psi$ implies $M, \sigma \vDash \psi'$, for all traces σ such that $\sigma[0] = s$.
- $M, s \vDash \mathsf{E}(\psi \rightsquigarrow \psi') \Leftrightarrow M, \sigma \vDash \psi$ implies $M, \sigma \vDash \psi'$, for some traces σ such that $\sigma[0] = s$.

- $M, s \vDash \mathbf{O}(\psi \rightsquigarrow \psi') \Leftrightarrow$ for every $\sigma \in \mathcal{NT}$ such that $\sigma[0] = s$ we have that for every $i \geq 0$, $M, \sigma[i..] \vDash \psi$ implies $M, \sigma[i..] \vDash \psi'$.
- $M, s \vDash \mathbf{P}(\psi \rightsquigarrow \psi') \Leftrightarrow$ for some $\sigma \in \mathcal{NT}$ such that $\sigma[0] = s$ we have that for every $i \geq 0$, $M, \sigma[i..] \vDash \psi$ implies $M, \sigma[i..] \vDash \psi'$.
- $M, s \vDash \mathbf{R}(\psi \rightsquigarrow \psi') \Leftrightarrow$ for every trace σ such that $\sigma[0] = s$ we have that for every $i \geq 0$: if $s[i] \notin \mathcal{N}$, then $M, \sigma[i..] \vDash \psi$ implies $M, \sigma[i..] \vDash \psi'$.

The above satisfaction relation makes use of dCTL satisfaction for path formulas, whose definition is standard:

- $M, \sigma \vDash \mathsf{X}\varphi \Leftrightarrow M, \sigma[1] \vDash \varphi$.
- $M, \sigma \vDash \varphi \, \mathcal{U} \, \varphi' \Leftrightarrow$ there exists $j \geq 0$ such that $M, \sigma[j] \vDash \varphi'$ and for every $0 \leq k < j$, it holds that $M, \sigma[j] \vDash \varphi$.
- $M, \sigma \vDash \varphi \, \mathcal{W} \, \varphi' \Leftrightarrow$ either there exists $j \geq 0$ such that $M, \sigma[j] \vDash \varphi'$ and for every $0 \leq k < j$ it holds that $M, \sigma[j] \vDash \varphi$, or for every $j \geq 0$ we have that $M, \sigma[j] \vDash \varphi$.

As usual, we will 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. We will often employ the shorthand $\mathbf{O}(\psi)$, meaning $\mathbf{O}(\top \rightsquigarrow \psi)$ (similarly for other operators and quantifiers). We also apply path operators to state formulas (we just used $\top$ in $\mathbf{O}(\top \rightsquigarrow \psi)$ as a path formula). This can be done thanks to the fact that every state formula φ can be expressed as a path formula, by $\bot \, \mathcal{U} \, \varphi$.

The above introduced deontic operators enjoy some useful properties, some of which we enumerate below. In the following properties, we use φ and φ' for state formulas and ψ for path formulas:

1. $\mathbf{O}(\bot) \equiv \mathbf{O}(\psi) \wedge \mathbf{O}(\neg\psi)$, where $\neg\psi$ denotes the negation of ψ, obtained using the dual temporal operators of ψ and pushing the negation inwards.
2. $\mathbf{O}(\top) \equiv \top$
3. $\mathbf{P}(\bot) \equiv \bot$
4. $\mathbf{R}(\bot) \vdash \mathsf{AG}\varphi \leftrightarrow \mathbf{O}(\varphi)$
5. $\mathbf{R}(\bot) \vdash \mathsf{EG}\varphi \leftrightarrow \mathbf{P}(\varphi)$
6. $\mathbf{R}(\top) \equiv \top$
7. $\mathbf{R}(\bot) \to \mathbf{P}(\top)$
8. $\mathbf{O}(\varphi) \wedge \mathbf{O}(\varphi') \to \mathbf{O}(\varphi \wedge \varphi')$
9. $\mathbf{O}(\varphi) \vee \mathbf{O}(\varphi') \to \mathbf{O}(\varphi \vee \varphi')$
10. $\mathbf{P}(\varphi \wedge \varphi') \to \mathbf{P}(\varphi) \wedge \mathbf{P}(\varphi')$
11. $\mathbf{P}(\varphi) \vee \mathbf{P}(\varphi') \to \mathbf{P}(\varphi \vee \varphi')$
12. $\mathbf{R}(\varphi) \wedge \mathbf{R}(\varphi') \to \mathbf{R}(\varphi \wedge \varphi')$
13. $\mathbf{R}(\varphi) \vee \mathbf{R}(\varphi') \to \mathbf{R}(\varphi \vee \varphi')$

Let us briefly explain these properties. Property 1 states that expressing that false is obliged (which is equivalent to saying that there will eventually be a fault) is the same as having contradicting obligations. Property 2 expresses that saying that true is obliged is equivalent to true. Similar properties hold for the permission operator. Property 3 indicates that false cannot be allowed. The deduction rules state that, in the absence of faults, the deontic operators can be expressed

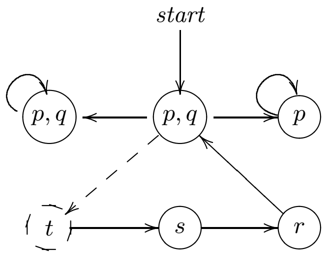


Fig. 1. A simple colored Kripke structure

using standard CTL. The properties of the operator $\mathbf{R}$ state that true always holds after a fault, while $\mathbf{R}(\bot)$ expresses that there will be no further faults in the future; this last expression implies $\mathbf{P}(\top)$, i.e., that there exist some good executions. Properties 8-13 relate the deontic operators to the standard boolean connectives. Due to space restrictions, we are unable to include the proofs of these properties in this paper; most of them can be proved straightforwardly resorting to the semantics of the involved operators.

In order to illustrate the semantics of the deontic operators, consider the colored Kripke structure in Figure 1, where the set of involved propositional variables is $\{p, q, r, s, t\}$, and each state is labeled by the set of propositional variables that hold in it. Also, the states that are the target of dashed arcs are abnormal states, also dashed, while the remaining ones are normal (i.e., dashed arcs are used for denoting transitions to faulty states, and the only faulty state in this model is the one labeled with t). It is obvious then that in every state of normal paths from the state indicated with *start*, p holds, which in dCTL is expressed as $\mathbf{O}(p)$. Also, there exist normal executions for which $p \wedge q$ always holds, expressed in dCTL as $\mathbf{P}(p \wedge q)$. On the other hand, the repair operator enables us to express properties regarding faulty states, and therefore also faulty executions. For instance, we can express that, immediately after every reachable fault, t holds, and a state in which r holds can be reached. In dCTL, these properties can be expressed as $\mathbf{R}(t)$ and $\mathbf{R}(\mathsf{F}r)$, respectively.

Finally, notice that other deontic operators, especially the *prohibition*, can be expressed using the above introduced ones. Prohibition can be characterized as $\mathbf{F}(\psi) = \neg\mathbf{P}(\psi)$. Intuitively, a (trace) property is forbidden when it cannot be true in a normal behavior. In other words, if such a property is continuously true in a trace, this trace contains some faults.

4 Fault Tolerance Reasoning in dCTL

Now that we have introduced our logic, let us start describing its use for expressing properties of systems in which faults might occur. We will illustrate the use of the logic using a few examples of typical fault tolerance situations.

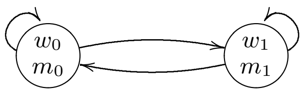


Fig. 2. A simple model of a memory cell, without faults

4.1 A Memory Cell

Let us consider a system composed of a simple memory cell, which stores a bit of information and supports reading and writing operations. Such a simple system can be characterized as the Kripke structure shown in Figure 2, where each state maintains the current value of the memory cell (m_i, for $i = 0, 1$) and the last write operation that was performed (w_i, for $i = 0, 1$). Obviously, in this system the result of a reading depends on the value stored in the cell. Thus, a property that one might associate with this model, is that the value read from the cell coincides with that of the last writing performed in the system. This property can straightforwardly be expressed using CTL, as follows: $\mathsf{AG}((m_0 \rightarrow w_0) \wedge (m_1 \rightarrow w_1))$. This can be considered part of the *requirements specification* that the implementation described in Fig. 2 is expected to satisfy. Of course, this system expresses ideal behavior and does not take into account faults of any kind, making the use of the deontic operators unnecessary. So let us consider some faults in this scenario. Suppose that, when a bit's value is 1, it can unexpectedly lose its charge and turn into a 0. In this case, the above implementation cannot guarantee the specification is satisfied, since it is obvious that, if after writing 1 in the cell the described fault occurs and a reading is performed, a 0 will be read instead of the last written value 1.

So, the above model must be altered in order to cope with the possibility of the described fault occurring. A typical mechanism for dealing with this situation in fault tolerance is via *redundancy*. For instance, one might decide to implement the same system now using three memory bits instead of one. Writing operations are performed simultaneously in the three bits, whereas reading operations will return the value that is repeated at least twice in the memory bits (known as voting), and write it back in all three of them. The resulting system is depicted in Figure 3. Each state in this model is described by a variable w_i which records the last writing operation performed, together with three bits, described by boolean variables c_0, c_1 and c_2. The occurrence of a fault, which changes a bit with value 1 to hold a 0, is represented by a dashed line. Faulty states (indicated by dashed circles) are those resulting from a fault occurrence. Standard continuous lines denote normal transitions between states, representing reading or writing operations.

Notice that the reading operation is defined in a different way, in the presence of redundancy; the read value is the one that is repeated the most, so reading a 1 can be logically expressed as $r_1 = (c_0 \wedge c_1) \vee (c_0 \wedge c_2) \vee (c_1 \wedge c_2)$. That is, the value read is a 1 if there are at least two "one bits" in the memory cell with redundancy. With r_1 defined, r_0 is defined simply as its negation.

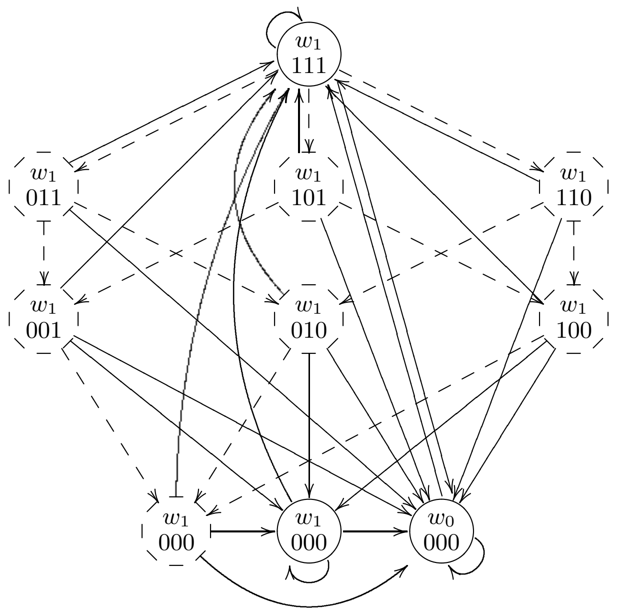


Fig. 3. A model of a memory cell, augmented with redundancy to deal with faults

Now let us discuss the properties one might expect of this augmented memory cell model. The original requirements of our model need to be updated to refer to our new implementation for reading a value (m_i is replaced by the above defined r_i): $\mathsf{AG}((r_0 \to w_0) \wedge (r_1 \to w_1))$. Of course, this property does not hold in the model, if faults occur. Still, this property is useful, since its verification, e.g. via model checking, would produce counterexamples that help us understand the situations in which our requirements are violated, in scenarios involving faults.

Besides the previous property, one of the most obvious properties one might be interested in is that, as long as no faults occur, the specification is guaranteed to hold. This can be thought of as a verification that the fault tolerance mechanism incorporated into to original system does not affect the satisfaction of the requirements specification when no faults occur. This first example of a fault tolerance property can be expressed naturally using obligation, in the following way:

$$\mathbf{O}((r_0 \to w_0) \wedge (r_1 \to w_1))$$

Let us start expressing properties of faulty scenarios. The motivation for introducing fault tolerance mechanisms is to be able to maintain the system behaving correctly even in the presence of faults. Of course, not every faulty scenario will maintain correct system behavior, so the general invariant property that we originally had becomes a *conditional* invariant, asserting that it will hold as long as fault occurrence is constrained. For example, for our memory cell with redundancy, we could say that the read value will coincide with the last written value even in the presence of faults, but as long as, whenever a fault occurs, no further faults happen before a read or write operation is performed. In dCTL, this is expressed as follows as follows:

$$\mathbf{R}((\mathsf{not\text{-}too\text{-}broken}\,\mathcal{U}\ \mathsf{bits\text{-}coincide}) \rightsquigarrow (r_i \rightarrow w_i))$$

where $\mathsf{not\text{-}too\text{-}broken} = \mathbf{P}(\top) \vee r_1$ (at most one fault has occurred since last read/write), and $\mathsf{bits\text{-}coincide} = (c_0 \leftrightarrow c_1) \wedge (c_0 \leftrightarrow c_2)$ (capturing that the three bits coincide, always a consequence of a read or write). Notice that the subformula to the right of $\rightsquigarrow$ is not restricted to normal behaviors, since it neither uses obligation nor permission operators. But the subformula to the left of $\rightsquigarrow$ restricts what must happen when faults occur, as we wanted: whenever a fault occurs the system must transit nonfaulty transitions, until a read or write operation is performed. This formula is an example of the use of the repair operation. It also employs $\mathbf{P}(\top)$, which expresses that the current state is a normal one (recall the restriction of colored Kripke structures that says that normal states must have at least a normal successor).

The pattern $\mathbf{R}(\psi \rightsquigarrow \phi)$ is a useful one in fault tolerance settings: it expresses that the state property ϕ is guaranteed to hold even in the presence of faults, as long as whenever a fault occurs, the system behaves as ψ indicates. Various interesting engineering questions arise in relation to this pattern (which we do not deal with in this paper); for instance, given a state formula ϕ, one might be interested in synthesizing the weakest formula ψ such that the previous pattern formula holds.

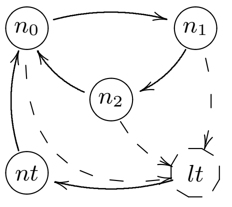
Another interesting property that dCTL enables us to express is that, regardless of how many consecutive faults occur, if the system is in a faulty state, i.e., a fault has just occurred (maybe immediately after another fault), the system always has the chance to behave in a way that the requirements of the system are reestablished. This is naturally expressed using permission, as follows:

$$\mathsf{AG}(\neg\mathbf{P}(\top) \wedge \mathsf{EX}\mathbf{P}(\top) \rightsquigarrow \mathsf{EG}\phi)$$

Since our model does not have unrecoverable faulty states, the formula $\neg\mathbf{P}(\top) \wedge \mathsf{EX}\mathbf{P}(\top)$ captures the property of a state being faulty (it is very easy to capture faulty states even in the presence of unrecoverable ones). Notice that the above "error avoidance" property holds in our model, since write operations always take us back to a state in which the requirements of the system are reestablished.

4.2 A Token Ring Protocol

Let us now consider another example. Suppose that we have a simple system composed of three connected nodes, whose activities are regulated via a token ring protocol. In an original system, the three nodes are connected in a ring topology, and a token is passed through by the nodes so that the node that has it in a particular time is the one with access to a particular resource, e.g., permission to send information across the network. It is not difficult to think of a few examples of properties that might be thought of as the requirements



n_i: Node i has the token
lt: Lost token
nt: New token created

Fig. 4. A model of a token ring of nodes, where tokens can be lost

of the system, such as there is always exactly one node who has the token, and whenever a node has a token, it eventually passes it to the next one in the ring.

A simple fault that can be conceived in this context is one in which, due to the unreliability of the medium, the token might be lost when being transmitted from a node to the next one. If the period that each node has the token is fixed, then a fault detection mechanism can be easily implemented using a timeout (if a node have not seen the token for more than the time limit for each node, times the number of nodes). An abstraction of this situation, including the fault detection and a recovery approach, is depicted in Figure 4. The states n_i correspond to the token being held by node i; when the token is lost, no node has it, and when the detection of the missing token is established, a new token is created, and given to node 0.

The requirements on this system can be straightforwardly specified using CTL, as follows:

$$\mathsf{AG}((n_0 \wedge \neg n_1 \wedge \neg n_2) \vee (\neg n_0 \wedge n_1 \wedge \neg n_2) \vee (\neg n_0 \wedge \neg n_1 \wedge n_2))$$
$$\mathsf{AG}(n_i \rightarrow \mathsf{AX}(n_{i \oplus 1}))$$

where $\oplus$ is addition modulo three. Notice that, for the sake of simplicity, we assume that each node has the token for exactly one instant of time (in the next step, the token has to belong to the next node). If one wants to check that these properties hold when no faults occur in the system, that can be expressed (and later on verified) using dCTL, in a similar way as for our previous example, i.e., by using obligation:

$$\mathbf{O}((n_0 \wedge \neg n_1 \wedge \neg n_2) \vee (\neg n_0 \wedge n_1 \wedge \neg n_2) \vee (\neg n_0 \wedge \neg n_1 \wedge n_2))$$
$$\mathbf{O}(n_i \rightsquigarrow \mathsf{AX}(n_{i \oplus 1}))$$

The second of the above original requirements, though guaranteed when no faults occur, fails in any scenario in which at least a fault occurs. This is a case in which a desirable property needs to be *relaxed*, rather than given up, due to faults: the requirement that the token must be passed to the next node in the *next* instant is transformed into the token being passed at *some future* moment to the next node. That is, the second of our requirements is relaxed into the following: $\mathsf{AG}(n_i \rightarrow \mathsf{AF}(n_{i \oplus 1}))$. This is a progress property that holds for the system, provided it behaves in a strongly fair fashion. Notice that, even though strong fairness in not expressible in CTL, this constraint is typically incorporated by various model checkers for the verification of liveness properties, and our property is a progress one, a particular case of liveness.

Another interesting fault tolerance property is one that expresses that faults are the only responsible of the token being lost. In other words, if the token is held by a particular node n_i, the token will be passed to the next node, or a fault will occur. This is expressed in dCTL in the following way:

$$\mathsf{AG}(n_i \to \mathsf{X}(\neg \mathbf{P}(\top) \vee n_{i\oplus 1}))$$

With the two simple case studies presented in this section, we tried to show examples of common properties of interest in the context of fault tolerant systems that can be, in our opinion, naturally expressed in dCTL. Other more general properties of fault tolerant systems, such as the concepts of *closure* and *convergence*, as described in [1], can also be expressed in a direct way. Closure serves the purpose of expressing that, given a state formula ϕ characterizing a required property of a system, φ is (inductively) preserved by the system by non-faulty transitions. In dCTL, this is expressed as $\mathbf{O}(\phi \rightsquigarrow \mathsf{AX}(\varphi))$. Convergence, on the other hand, allows one to express that, from any state satisfying certain property ϕ' (e.g., indicating that the system might be "'mildy" broken), if no further faults occur then the system always comes back to a state satisfying φ (i.e., it eventually recovers from the fault). This can be expressed separated in two parts. First, we express that from any normal state satisfying φ', if we move through nonfaulty transitions, we can eventually reach a state in which φ holds: $\mathbf{O}(\varphi' \rightsquigarrow \mathsf{AF}(\varphi))$. Second, we say that whenever a fault occurs, if we move through states that are normal or satisfy φ', then we can eventually reach a state in which φ holds: $\mathbf{R}(\mathsf{G}(\varphi' \vee \mathbf{P}(\top)) \rightsquigarrow \mathsf{F}\varphi)$. Some of the properties we dealt with in our examples can be thought of as variants of these two concepts.

5 Expressivity and Complexity of dCTL

In this section we show some results regarding the expressiveness and complexity of our logic dCTL. The complexity results enable us to show not only that the above properties of fault tolerant systems can be automatically checked, but also that checking them can be done in polynomial time with respect to the sizes of the model and the verified formula. We start by showing that dCTL formulas can be model checked, by providing a characterization of our logic into the more expressive logic CTL*. This characterization, which is not difficult to devise, introduces a fresh propositional letter n in the encoding, to "mark" normal behaviors. This translation is formalized in the following definition.

Definition 1. *The translation τ from dCTL formulas over an alphabet AP, to CTL* formulas over the alphabet $AP \cup \{n\}$, for some symbol $n \notin AP$, is defined as follows:*

- $\tau(\top) = \top$.
- $\tau(p_i) = p_i$.
- $\tau(\neg\varphi) = \neg\tau(\varphi)$.

- $\tau(\varphi \to \varphi') = \tau(\varphi) \to \tau(\varphi')$.
- $\tau(\mathsf{A}(\psi \rightsquigarrow \psi')) = \mathsf{A}(\tau(\psi) \to \tau(\psi')))$.
- $\tau(\mathsf{E}(\psi \rightsquigarrow \psi')) = \mathsf{E}(\tau(\psi) \to \tau(\psi'))$.
- $\tau(\mathbf{O}(\psi \rightsquigarrow \psi')) = \mathsf{A}(\mathsf{G}n \to \mathsf{G}(\tau(\psi) \to \tau(\psi')))$
- $\tau(\mathbf{P}(\psi \rightsquigarrow \psi')) = \mathsf{E}(\mathsf{G}n \wedge \mathsf{G}(\tau(\psi) \to \tau(\psi')))$
- $\tau(\mathbf{R}(\psi \rightsquigarrow \psi')) = \mathsf{A}(\mathsf{G}(\neg n \to (\tau(\psi) \to \tau(\psi'))))$
- $\tau(\mathsf{X}\varphi) = \mathsf{X}(\tau(\varphi))$.
- $\tau(\varphi \,\mathcal{U}\, \varphi') = \tau(\varphi)\,\mathcal{U}\,\tau(\varphi')$.
- $\tau(\varphi \,\mathcal{W}\, \varphi') = \tau(\varphi)\,\mathcal{W}\,\tau(\varphi')$.

The above translation from dCTL to CTL^* is semantics preserving. The following mapping between Kripke structures and colored Kripke structures enables us to argue about the semantics preservation.

Definition 2. *Let $M = \langle S, R, L\rangle$ be a Kripke structure defined over an alphabet $AP \cup \{n\}$. From M, we define the colored Kripke structure $M^* = \langle S, R, L', \mathcal{N}\rangle$ over the alphabet AP, in the following way:*

- *L' is L restricted to AP.*
- *$s \in \mathcal{N} \Leftrightarrow M, s \vDash n$.*

The following theorem shows that our embedding of dCTL in CTL^* is semantics preserving.

Theorem 1. *For every $M = \langle S, R, L\rangle$ defined over an alphabet $AP \cup \{n\}$, and every dCTL formula φ over AP, the following holds:*

$$M^* \vDash_{dCTL} \varphi \Leftrightarrow M \vDash_{CTL^*} \tau(\varphi).$$

Proof. By induction on the structure of formulas.

It is worth noting that our translation of dCTL deontic operators to CTL^* involves some CTL^* formulas which are not expressible in CTL. In particular, the formula $\neg\mathbf{P}(p \rightsquigarrow \mathsf{X}p)$, which is translated to $\mathsf{A}(\mathsf{F}\neg n \vee \mathsf{F}(p \wedge \mathsf{X}\neg p))$ in CTL^*, is not expressible in CTL, nor in none of its extensions CTL^+, ECTL and ECTL^+. This expressiveness result, which follows from properties given in [9], is summarized in the following theorem.

Theorem 2. *The expressive powers of logics CTL, dCTL, CTL^+, ECTL, ECTL^+ and CTL^*, are related by the following diagram of inclusions:*

$$\begin{tikzcd}
 & \mathit{dCTL} \arrow[rrd] & & \\
\mathit{CTL} \arrow[ru] \arrow[r] \arrow[rd] & \mathsf{CTL}^+ \arrow[r] & \mathsf{ECTL}^+ \arrow[r] & \mathsf{CTL}^* \\
 & \mathit{ECTL} \arrow[ru] \arrow[rru] & &
\end{tikzcd}$$

Proof. A proof of $\mathsf{AF}(p \wedge \mathsf{X}\neg p)$ not being expressible in ECTL^+ can be found in [9] (cf. Theorem 5 therein). We can use a similar argument to prove that $\mathsf{A}(\mathsf{F}(\neg n) \vee \mathsf{F}(p \wedge \mathsf{X}\neg p))$ is not expressible in ECTL^+ either. Consider the sequence of models $N_1, N_2, N_3, \ldots$ and $M_1, M_2, M_3, \ldots$ used in Theorem 5 of [9], and set n to true in every state of these models. Since we have that $M_i, a_i \vDash \mathsf{AF}(p \wedge \mathsf{X}\neg p)$, we also have that $M_i, a_i \vDash \mathsf{A}(\mathsf{F}(\neg n) \vee \mathsf{F}(p \wedge \mathsf{X}\neg p))$. Since it is also the case that $N_i, a_i \nvDash \mathsf{AF}(p \wedge \mathsf{X}\neg p)$, for every i, then it must be the case that $N_i, a_i \nvDash \mathsf{A}(\mathsf{F}(\neg n) \vee \mathsf{F}(p \wedge \mathsf{X}\neg p))$. Taking into account that these structures cannot be distinguished by ECTL^+ formulas, it is straightforward the fact that $\neg\mathbf{P}(p \to \mathsf{X}p)$ is not expressible in ECTL^+. Moreover, this formula is not expressible in any of the logics CTL, ECTL or CTL^+, which are sublogics of ECTL^+.

This same theorem can also be extended to prove that some dCTL formulas are not expressible in other related logics, particularly CTL^2. The argument for the proof is essentially the same used in the above proof.

The model checking problem for CTL, ECTL, CTL^+ and CTL^2 is in P, while for ECTL^+ and CTL^* this problem is PSPACE-complete. Our logic is more expressive than CTL and is able to express formulas not expressible in ECTL^+, so a natural concern is whether the model checking problem for our logic is PSPACE-complete, which would be an unwanted, but reasonable, price paid for its expressiveness. As we show below, the model checking problem for dCTL maintains a polynomial complexity. This, combined with the fact that dCTL is able to express properties not expressible in other known “efficient” (in the sense that their model checking is in P) sublogics of CTL^*, make our logic a novel fragment of CTL^*.

Theorem 3. *The model checking problem for dCTL is in P.*

Proof. The main idea behind the proof is the adaptation of the algorithm described in [2] for CTL model checking, to support also checking deontic formulas. These additional processes can be done in polynomial time, using reachability algorithms.

Temporal models are implemented as graphs, so our set $\mathcal{N}$ of colored Kripke structures can be captured simply by adding a boolean variable n, set to true in exactly those states that belong to $\mathcal{N}$ (recall that, according to our restriction on colored Kripke structures, if n is true in some state s, then n is true in some successor of s). In order to check $M \vDash \varphi$, we start by calculating the sets $Sat(\psi) = \{s \mid M, s \vDash \psi\}$, for every subformula ψ of φ, starting from the subformulas at the bottom in the syntax tree of φ. The main technical difficulty is avoiding the exponential blow up in the translation of formulas of the form $\mathsf{A}(\psi \rightsquigarrow \phi)$ and $\mathsf{E}(\psi \rightsquigarrow \phi)$, etc. This blow up is avoided since these quantifiers always apply to a boolean combination of at most two path formulas. These formulas can then be checked using the process for the equivalent formulas in CTL.

It remains to show how to check formulas of the form $\mathbf{O}(\psi \rightsquigarrow \psi')$ and $\mathbf{P}(\psi \rightsquigarrow \psi')$. For the sake of simplicity, and without loss of generality, we can restrict the analysis to deontic operators applied to a single path formula (it is known that implications of path formulas in the scope of a path quantifier can be translated to a state formula of a fixed length). Consider the formula $\mathbf{O}(\psi)$, which is equivalent to $\mathsf{A}(\mathsf{G}n \rightarrow \mathsf{G}(\psi))$. In order to build the set $Sat(\mathbf{O}(\psi))$, we can restrict the building process to states where n is true, and calculate that this set of states satisfies $\mathsf{A}(\psi)$. This can simply be checked using the model checking algorithm for CTL. Now consider $\mathbf{P}(\psi)$. This formula is equivalent to the CTL^* formula $\mathsf{E}(\mathsf{G}n \wedge \mathsf{G}\psi)$. In order to build the set $Sat(\mathbf{P}(\psi))$, we check that there exists some path of states satisfying n where $\mathsf{G}\psi$ is true; this is done by checking $\mathsf{E}\psi$ for the nodes satisfying n (this can be done in polynomial time, inductively). Then, $s \in Sat(\mathbf{P}(\psi))$ if there is some successor of these states which satisfies both n and $\mathsf{E}\psi$. Finally, checking $\mathbf{R}(\psi)$ demands a similar technique. In summary, these processes can be performed using a depth-first search, and the algorithms for checking CTL formulas. Our extra checking processes are polynomial, therefore the final model checking algorithm is also polynomial with respect to the size of the model and the length of the formula.

6 Conclusions and Future Work

We have proposed a computation tree logic especially tailored for describing temporal properties of fault tolerant systems, and employing temporal deontic operators for this purpose. The deontic operators, which help in making a distinction between normal and abnormal states and behaviors, provide an expressiveness that is sufficiently rich for describing various properties of interest in the context of fault tolerance. We showed that some formulas expressible in our logic cannot be expressed in other known fragments of CTL^*, including ECTL^+ and its sub-logics. However, and as opposed to the case for ECTL^* and CTL^*, for which the model checking problem is PSPACE-complete, model checking our logic dCTL is in P.

These results, together with our arguments regarding the usefulness of the logic for fault tolerance system specification, make it an interesting fragment of CTL^*. In order to argue about its usefulness, we have developed two small case studies of fault tolerance situations, which despite their simplicity enabled us to illustrate the expressivity of the logic. Expressing temporal properties regarding fault tolerance could alternatively be achieved by a more "low level" approach, e.g., directly referring to faulty states via some atomic state formula capturing exactly such states. We believe that our deontic operators provide an indirect, higher level, way of referring to faults in the expression of fault tolerance properties, capturing some patterns useful in this context. Moreover, properties of deontic operators allow one to reason about formal descriptions at a higher level of abstraction.

We are currently exploring various lines of future work. We are developing more complex examples, and we are experimenting with the use of a μ-calculus model checker, Mucke, used as a target to express dCTL formulas. We are also analyzing alternative deontic operators that would provide an expressive power equivalent to that of our current version of the logic, but featuring a more intuitive reading. Also, we have not been concerned so far about providing an actual formalism in which the system, the associated faults and the fault tolerance mechanisms are described, in a methodologically sound way. We plan to develop such a setting, incorporating our logic in it.

Acknowledgements. The authors would like to thank Pedro D'Argenio and the anonymous referees for their valuable comments. This work was partially supported by the Argentinian Agency for Scientific and Technological Promotion (ANPCyT), through grant PICT PAE 2007 No. 2772. The fourth author's participation was also supported through ANPCyT grant PICT 2006 No. 2484.

References

1. Arora, A., Gouda, M.: Closure and Convergence: A Foundation of Fault-Tolerant Computing. IEEE Transactions on Software Engineering 19(11) (1999)
2. Baier, C., Katoen, J.-P.: Principles of Model Checking. The MIT Press, Cambridge (2008)
3. Castro, P., Maibaum, T.: Deontic Action Logic, Atomic Boolean Algebras and Fault-Tolerance. Journal of Applied Logic 7(4) (2009)
4. Clarke, E., Draghicescu, I.: Expressibility Results for Linear Time and Branching Time Logic. In: de Bakker, J.W., de Roever, W.-P., Rozenberg, G. (eds.) Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. LNCS, vol. 354, pp. 428–437. Springer, Heidelberg (1989)
5. Clarke, E., Emerson, E., Sistla, A.: Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications. ACM Transactions on Programming Languages and Systems 8(2) (1986)
6. Clarke, E., Grumberg, O., Peled, D.: Model Checking. The MIT Press, Cambridge (1999)
7. Coenen, J.: Specifying Fault Tolerant Programs in Deontic Logic, Computing Science Notes 91/34, Dept. of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands (1991)
8. D'Ippolito, N., Braberman, V., Piterman, N., Uchitel, S.: Synthesis of Live Behaviour Models for Fallible Domains. In: Proc. of International Conference on Software Engineering ICSE 2011. IEEE Press, Los Alamitos (2011)
9. Emerson, E., Halpern, J.: "Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic. J. ACM 33(1) (1986)
10. French, T., McCabe-Dansted, J., Reynolds, M.: A Temporal Logic of Robustness. In: Konev, B., Wolter, F. (eds.) FroCos 2007. LNCS (LNAI), vol. 4720, pp. 193–205. Springer, Heidelberg (2007)
11. Gnesi, E., Lenzini, G., Martinelli, F.: Logical Specification and Analysis of Fault Tolerant Systems through Partial Model Checking. Electronic Notes on Theoretical Computer Science, vol. 118. Elsevier, Amsterdam (2005)
12. Janowski, T.: On Bisimulation, Fault-Monotonicity and Provable Fault-Tolerance. In: Johnson, M. (ed.) AMAST 1997. LNCS, vol. 1349, pp. 292–306. Springer, Heidelberg (1997)
13. Magee, J., Maibaum, T.: Towards Specification, Modelling and Analysis of Fault Tolerance in Self Managed Systems. In: Proc. of International Workshop on Self-Adaptation and Self-Managing Systems SEAMS 2006. ACM Press, New York (2006)
14. McCabe-Dansted, J., French, T., Reynolds, M., Pinchinat, S.: On the Expressivity of RoCTL*. In: Proc. of the 16th International Symposium on Temporal Representation and Reasoning TIME 2009. IEEE Computer Society, Los Alamitos (2009)