Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 258 (2009) 17–34

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Deontic Logic, Contrary to Duty Reasoning and Fault Tolerance[3]

Pablo F. Castro[1] T.S.E. Maibaum[2]

Department of Computing and Software
McMaster University
Hamilton, Canada

Abstract

Deontic Logic was introduced in the first half of the last century to formalize aspects of legal reasoning. Since then a lot of effort has gone into improving the formalism(s) and widening their applicability, including in Computer Science and Software Engineering. One strand of work has focused on the use of an action based approach to deontic operators, rather than the traditional property focused operators. We propose a new version of this kind of deontic logic that has very nice meta-logical properties, avoids many of the traditional problems of deontic logics and has an appealing treatment of contrary to duty reasoning. This kind of reasoning provides a kind of conditional reasoning about having violated normative constraints and describing the resulting consequences. We show how to apply this formalism to characterize fault tolerance mechanisms and to then reason about the properties of the mechanisms.

Keywords: Fault-Tolerance, Deontic Logic, Software Specification, Software Verification

1 Introduction

Deontic logic is a branch of modal logic which focuses on the study of the reasoning arising in ethical and moral contexts, which usually involve norms and prescriptions (see [3,12]). These logics, usually, consider two new modalities: *permission* and *obligation*. Of course, related to them are the concepts of *prohibition* and *violation*. These concepts arise naturally in fault-tolerance, where some actions are "ideal" or "obligatory", and the execution of any other action yields an error state. We have proposed a deontic logic in [11], and in [9] we have studied the application of this logic to fault-tolerance.

In fault-tolerant systems, it is usual to have situations where, after a violation, we must perform some actions to recover from this violation. This is an instance of

[1] Email:castropf@mcmaster.ca

[2] Email: tom@maibaum.org

[3] This work was supported by the Natural Sciences and Engineering Research Council of Canada and the Department of Computing and Software and the Faculty of Engineering, McMaster University.

doi:10.1016/j.entcs.2009.12.011

what is called by deontic logicians *contrary-to-duty* (or CTD for short) reasoning. CTD reasoning has been an important object of study in deontic logic; this kind of reasoning arises naturally in legal scenarios. However, in several deontic logics CTD scenarios are inconsistent when formalized; this is sometimes thought of as being *paradoxical*, since this is contrary to our intuition (in the sense that, intuitively, these statements are not inconsistent).

In this paper we extend the logics introduced in [11] with the goal in mind of obtaining a more expressive framework for allowing us to avoid the classical problems that arise in contrary-to-duty scenarios. Furthermore, we present some examples to illustrate that CTD statements are common in specification of fault-tolerant systems, and we show how we can deal with them using the proposed logic.

The paper is organized as follows. In section 2 we give a brief introduction to deontic logic and contrary-to-duty statements. In section 3 we describe our version of deontic logic and we propose a more expressive extension. In sections 4 and 5 we show two examples: a simple train system and a formalization of the Byzantine Generals problem [18]. These case studies allow us to demonstrate that CTD reasoning arises naturally in fault-tolerance. We prove some properties of the examples to illustrate the benefits of using deontic logic to specify and verify faul-tolerant systems.

2 Deontic Logics

Ernst Mally was the first to try to capture the reasoning underlying norms and prescriptions using a formal system, in particular Mally introduced obligation as a predicate (together with other related operators) and provided an axiomatic system for his logic. However, in Mally's logic the concept of obligation is superfluous, in the sense that, if we take $\mathsf{O}(\varphi)$ as saying it ought to be the case that φ is true, then we obtain $\varphi \to \mathsf{O}(\varphi)$, which trivializes the deontic operators. Since then, several deontic logics have been proposed in the literature. Perhaps the most studied is the so-called *Standard Deontic Logic* (or SDL) [12]. SDL is a particular case of normal modal logics; this logic has the modality $\mathsf{O}(\varphi)$ (φ is obligatory); and the following axioms are proposed to capture the notion of obligation [20]:

SDL0. all the tautologies of the language.

SDL1. $\mathsf{O}(\varphi \to \psi) \to (\mathsf{O}(\varphi) \to \mathsf{O}(\psi))$.

SDL2. $\mathsf{O}(\varphi) \to \neg\mathsf{O}(\neg\varphi)$.

For the rules we have:

- If $\vdash \varphi \to \psi$ and $\vdash \varphi$, then $\vdash \psi$,
- If $\vdash \varphi$, then $\vdash \mathsf{O}(\varphi)$.

Equivalent axiomatizations of SDL can be found in [12] (this system is called $\mathbf{OK}^+$ in [3]). The second deduction rule means that we have a normal modal system.

The semantics of SDL is given with Kripke structures [5], and the interpretation of the obligation operator is the same as the ussual modal necessity (although this axiomatization imposes a different structure on the Kripke models; note that the axioms imply that the Kripke structures are *serial*, i.e., every state has a successor).

The intuition of the semantics is as follows. If a state w is related with another state v, this mean that the obligations occurring in w are true in v (in some way we can think of v as an "ideal" world for w). Several consequences of this axiomatic system have been criticized for being contrary to the intuitive properties of obligation. For example, a consequence of these axioms is the property $\mathsf{O}(\top)$ (which can be read as saying that there are always obligations; at least we have that all the tautologies are obliged). Some people have argued that there could be scenarios where nothing is obliged, and these kinds of scenarios are not possible in SDL. Another problematic issue is the definition of permission which is introduced in the logic as a dual of obligation, that is: $\mathsf{P}(\varphi) \leftrightarrow \neg\mathsf{O}(\neg\varphi)$. Note that this definition together with axiom **SDL2**, imply that we have the following theorem: $\mathsf{O}(\varphi) \to \mathsf{P}(\varphi)$, i.e., obligation implies permission. If we see permission as modal possibility (which is the case in SDL), then we have what is sometimes called Kant's law: obligation implies possibility (i.e., $\mathsf{O}(\varphi) \to \Diamond\varphi$). It is not hard to find examples where this property is not desirable. It is not our intention to defend or argue against this logical system; readers can take their own position. Further discussion about these topics can be found in [12].

As explained in the introduction, we are interested in contrary-to-duty statements. Let us introduce a standard example of CTD statements, the so-called *Gentle Killer* paradox [14]. It can be stated as follows:

- It is forbidden to kill.
- If you kill, you have to kill gently.
- You kill.

In SDL the formalization of this paradox gives us an inconsistent set of sentences [24]. The main problem is that, in SDL, incompatible obligations are inconsistent, i.e., $\vdash \mathsf{O}(\varphi) \wedge \mathsf{O}(\neg\varphi) \to \bot$. Contrary-to-duty scenarios are also usual in fault-tolerance. We illustrate this fact with two examples in sections 4 and 5.

The Standard Deontic Logic is an *ought-to-be* deontic logic, i.e., the deontic predicates are applied to predicates (e.g., it is obligatory that it is raining). Many authors (e.g., [21,17]) have pointed out that several problems of deontic logics (paradoxical statements and non-intuitive properties) can be solved by applying deontic operators to actions instead of predicates. In [21], Meyer proposes to reduce deontic predicates to modal operators (in a dynamic logic setting [15]). The main idea behind this work is to use a constant predicate to indicate when a violation has occurred. In dynamic deontic logic, we have propositional formulae together with formulae of the type: $[\alpha]\varphi$, where α is an action and φ is a formula. Actions are built from a set of atomic actions and operators. Dynamic logics have the following operators: ; (sequential composition), $\sqcup$ (choice), * (iteration). Intuitively, the formula $[\alpha]\varphi$ means *after executing the action α, the formula φ becomes true.*

In other words, using these formulae we can express specifications of actions in a pre/post-condition style. Meyer reduces the notion of permission to modalities, as follows: $\mathsf{P}(\alpha) \equiv \langle\alpha\rangle\neg\mathsf{v}$, i.e., an action is allowed if and only if there is some way of executing it such that we get a state without violations (where v represents the idea that a violation has occured). A similar approach is presented by the FOREST project [17,19], where a modal logic with actions (called MAL) is introduced, but MAL considered several other action operators, for example: $\sqcap$ (parallel execution) and $-$ (action complement). Many other variations of deontic logic with actions have been presented in the literature. One difference between the logic presented in section 3 and Meyer's logic (and related approaches) is that in our logic there is no relationship between deontic predicates and standard modalities, i.e., we do not reduce deontic predicates to the concepts of possibility or necessity. We reject the reduction of deontic predicates to modalities since we think that prescription (what the system should do) and description (what the system does) must not be mixed up. For example, following Meyer's definition of permission, allowed actions are also recovery actions (they recover the system from an error state). This is undesirable in fault-tolerance: permitted actions may carry forward violations; this is also noted by Sergot in [23].

3 A Deontic Action Logic with Stratified Norms

We have presented a deontic action logic in [11,8,7,10]; in this section we review briefly the basic definitions of this logic and we introduce an extension for this logic to deal with contrary-to-duty reasoning.

We use *vocabulary* (or *language*) to refer to a tuple $L = \langle\Delta_0, \Phi_0, V_0, I_0\rangle$, where Δ_0 is a finite set of *primitive actions*: $a_1, ..., a_n$, which represent the possible actions of a part of the system and, perhaps, of its environment. Φ_0 is an enumerable set of propositional symbols denoted by $p_1, p_2, \ldots$. V_0 is a finite subset of $\mathcal{V}$, where $\mathcal{V} = \{\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3, \ldots\}$ is an infinite, enumerable set of "violation" propositions. The indices in I_0 correspond to a stratification of the concept of norm, where the stratification corresponds to degrees of fault in the system being modeled. All these sets are mutually disjoint. The atomic actions in a vocabulary can be combined as follows to form more complex action terms (denoted by Δ): $\Delta_0 \subseteq \Delta$ and $\emptyset, \mathbf{U} \in \Delta$. If $\alpha, \beta \in \Delta$, then $\alpha \sqcup \beta \in \Delta$, $\alpha \sqcap \beta \in \Delta$ and $\overline{\alpha} \in \Delta$. No other expression belongs to Δ. On the other hand, the formulae of this logic (denoted by Φ) are defined as follows. If $\varphi \in \Phi_0 \cup V_0$, then $\varphi \in \Phi$. If φ and ψ are formulae, then $\psi \to \psi, \neg\psi \in \Phi$. If φ is a formula and α an action, then $[\alpha]\varphi$ is a formula. If α is an action and $i \in I_0$, then $\mathsf{P}^i_{\mathsf{w}}(\alpha)$ and $\mathsf{P}^i(\alpha)$ are formulae. If φ and ψ are formulae, then $\mathsf{EN}\varphi$, $\mathsf{A}(\varphi\,\mathcal{U}\,\psi)$ and $\mathsf{E}(\varphi\,\mathcal{U}\,\psi) \in \Phi$. If α and β are actions and $S \subseteq \Delta_0$, then $\mathsf{Done}_S(\alpha)$ and $\alpha =_{act} \beta$ are formulae. B is a formula.

The intuitive reading of the modal and deontic formulae is as follows:

- $[\alpha]\varphi$, *after executing α in any possible way, φ will hold.*
- $[\alpha_1 \sqcup \alpha_2]\varphi$, *every way of executing α_1 or α_2 leads to φ.*

- $[\overline{\alpha}]\varphi$, *after executing an action different from* α, φ *holds.*
- $[\alpha_1 \sqcap \alpha_2]\varphi$, *every way of executing both* α_1 *and* α_2 *leads to* φ.
- $\mathsf{P}^i(\alpha)$, *all the different ways of executing* α *are allowed, for the stratified level* i.
- $\mathsf{P}^i_{\mathsf{w}}(\alpha)$, *there is at least one way of executing* α *which is allowed, for the stratified level* i.
- $[\mathbf{U}]\varphi$, *any execution of the component yields* φ.
- $\mathsf{EN}\varphi$, *in some path, in the next instant* φ *is true.*
- $\mathsf{A}(\varphi\, \mathcal{U}\, \psi)$, *in every path the formulae* φ *is true until* ψ *becomes true.*
- $\mathsf{E}(\varphi\, \mathcal{U}\, \psi)$, *in every path the formulae* φ *is true until* ψ *becomes true.*
- $\mathsf{Done}(\alpha)$, α *was the last action executed.*

B is true at the beginning of time and so denotes the initial state of execution of any system. The temporal operators have the standard meaning in a branching temporal logic. Note that, if we consider some complete axiomatization of boolean algebras Φ_{BA}, then we obtain Δ/Φ_{BA} the boolean (atomic) algebra of action terms. It is important to note that the atoms in this boolean algebra are each interpreted as singletons or the empty set.

Definition 3.1 [models] Given a language $L = \langle \Phi_0, \Delta_0, V_0, I_0 \rangle$, a *L-Structure* is a tuple: $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \{\mathcal{P}^i \mid i \in I_0\} \rangle$ where:

- $\mathcal{W}$, is a set of worlds.
- $\mathcal{E}$, is a non-empty set of (names of) events.
- $\mathcal{R}$, is an $\mathcal{E}$-labelled relation between worlds. We require that, if $(w, w', e) \in \mathcal{R}$ and $(w, w'', e) \in \mathcal{R}$, then $w' = w''$, i.e., $\mathcal{R}$ is functional.
- $\mathcal{I}$, is a function:
  · For every $p \in \Phi_0 : \mathcal{I}(p) \subseteq \mathcal{W}$
  · For every $\alpha \in \Delta_0 : \mathcal{I}(\alpha) \subseteq \mathcal{E}$, and $\mathcal{I}(\alpha)$ is finite.
  In addition, the interpretation $\mathcal{I}$ has to satisfy the following properties:
  I.1 For every $\alpha_i \in \Delta_0$: $|\mathcal{I}(\alpha_i) - \bigcup\{\mathcal{I}(\alpha_j) \mid \alpha_j \in (\Delta_0 - \{\alpha_i\})\}| \leq 1$.
  I.2 For every $e \in \mathcal{I}(a_1 \sqcup \ldots \sqcup a_n)$: if $e \in \mathcal{I}(\alpha_i) \cap \mathcal{I}(\alpha_j)$, where $\alpha_i \neq \alpha_j \in \Delta_0$, then: $\cap\{\mathcal{I}(\alpha_k) \mid \alpha_k \in \Delta_0 \wedge e \in \mathcal{I}(\alpha_k)\} = \{e\}$.
  I.3 $\mathcal{E} = \bigcup_{\alpha_i \in \Delta_0} \mathcal{I}(\alpha_i)$.
- each $\mathcal{P}^i \subseteq \mathcal{W} \times \mathcal{E}$ is a relation which indicates which event is permitted in which world with respect to permissions with index i.

Roughly speaking, the structure gives us a labelled transition system, whose labels are events, which are produced by some local action or they could also correspond to external events. Note that we have a set of events, but actions are only interpreted over finite subsets, whose intersections satisfy the Hausdorff condition (implied by conditions **I.1** and **I.2**), i.e., we require that every one-point set can be generated from the actions of the component; this ensures that the labels in the transitions are uniquely determined by some parallel execution of actions in the component and perhaps some environmental actions.

We use maximal traces to give the semantics of the temporal operators. Below, we use the following notation. Given an infinite trace (or path) $\pi = s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} ...$, we denote by $\pi^i = s_i \xrightarrow{e_i} s_{i+1} \xrightarrow{e_{i+1}} ...$ the subpath of π starting at position i. The notation $\pi_i = s_i$ is used to denote the i-th element in the path, and we write $\pi[i, j]$ (where $i \leq j$) for the subpath $s_i \xrightarrow{e_i} ... \xrightarrow{e_j} s_{j+1}$. Meanwhile $\pi(i)$ denotes the event e_i. Finally, given a finite path $\pi' = s'_0 \xrightarrow{e'_0} .... \xrightarrow{e'_n} s_{n+1}$, we say $\pi' \preceq \pi$ if π' is an initial subpath of π, that is: $s_i = s'_i$ and $e_i = e'_i$ for $0 \leq i \leq n$, and we denote by $\prec$ the strict version of $\preceq$.

The formal definition of the relationship $\pi, i, M \vDash \varphi$ (φ is true at instant i in the structure M) can be found in [6]. In [6] we present a sound and complete axiomatic system for this logic.

Having several versions of permissions is useful in practice, in particular when we have contrary-to-duty statements. Consider, for example, the *Gentle Killer* paradox (introduced in section 2):

- It is forbidden to kill.
- If you kill, you ought to kill gently.
- You kill.

As explained before, in SDL the formalization of this paradox gives us an inconsistent set of sentences [24]. We can formalize this scenario as follows:

- $\mathsf{F}^1(\mathtt{k})$
- $\mathtt{kg} \sqsubseteq \mathtt{k}$
- $\mathtt{nk} \sqcap \mathtt{k} =_{act} \emptyset$
- $\mathsf{O}^2(\mathtt{nk} \sqcup \mathtt{kg})$
- $\mathsf{ANDone}(\mathtt{k})$

We consider the actions $\mathtt{k}$ (kill), $\mathtt{kg}$ (kill gently) and $\mathtt{nk}$ (not kill). The first axiom says that it is forbidden to kill. The second formula says that the action of killing gently ($\mathtt{kg}$) is a way of killing. The third axiom expresses that killing and not killing are disjoint actions. The fourth formula says that, if we will kill, then we have to kill gently. In the last formula, we use the $\mathsf{Done}()$ operator to state that we will kill (which is expressed saying that the next action is to kill). In this case, we consider in the vocabulary two indexes: 1 and 2 pointing to the fact that there are two different levels of norms in the specification. In contrast to standard deontic logic, these sentences are not contradictory in our setting. For example, the structure illustrated in figure 1 is a model of this set of sentences. The structure in this figure has three states w, w_1, w_2, and we have $\mathcal{I}(\mathtt{k}) = \{e_1, e_2\}$, $\mathcal{I}(\mathtt{kg}) = \{e_1\}$ and $\mathcal{I}(\mathtt{nk}) = \{e_3\}$. The labels on the transitions indicate which actions are executed and which are not in each transition. The upper dashed arrow denotes a transition that is forbidden with respect to index 1 but not for index 2. The lower dashed arrow indicates an arrow which is forbidden for both indexes. In [22], Meyer sketched an extension of dynamic deontic logic where he considers several versions of permission and obligation; he uses this extension to model some paradoxes; however,

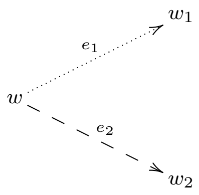

Fig. 1. Model for the Gentle Killer

in this version deontic predicates are reduced to modalities, and therefore, several violation predicates are introduced in the logic to model permission. As we explain above, defining permission by means of modalities is not always a good choice when modelling fault-tolerant systems: the notions of prescription and description are mixed up and, as we remarked above, this is not desirable in fault-tolerance, where the notion of allowed or permitted action must be different to that of recovery action.

4 First Example: A Simple Train System

We consider a simple example of a train system. Train systems are those systems that control the movement of trains through a network of rail segments. Fault-tolerance is a key aspect of these systems: a fault in the system may cause a train collision and the loss of human life. These kinds of systems are the object of active research in the fault-tolerance community, see [2,16,1].

Our system is made up of a collection of trains: $\mathtt{t_1}, \dots, \mathtt{t_n}$ and a set of rail segments $\mathtt{r_1}, \dots, \mathtt{r_m}$ (we assume $\mathtt{n} < \mathtt{m}$). Rail segments are connected to other rail segments; in each of these connections, the rails have one signal controlling access to them. The goal of the signal is to prevent trains from entering into a segment when another train is already in it. The signals can be green (when the segment is free) or red (when another train is in the segment). We have the following predicates. For each $0 \leq \mathtt{i} \leq \mathtt{n}$ and $0 \leq \mathtt{j} \leq \mathtt{m}$, we have a predicate $\mathtt{t_i.r_j}$ which is true when the train $\mathtt{t_i}$ is in segment $\mathtt{r_j}$; we also have a predicate $\mathtt{t_i.s}$ (true when the train is stopped). For each $0 \leq \mathtt{j} \leq \mathtt{m}$, we have a proposition $\mathtt{r_j.green}$ which is true when the signal of the rail $\mathtt{r_j}$ is green. We have a violation predicate $\mathsf{v_j}$ for every $0 \leq \mathtt{j} \leq \mathtt{m}$ which is true when we have two trains in the segment $\mathtt{r_j}$. (This is implemented by a sensor in the segment which detects the two trains.) Finally, we have propositions $\mathtt{r_i} R \mathtt{r_j}$ which indicate that $\mathtt{r_i}$ and $\mathtt{r_j}$ are connected.

We have the following actions: $\mathtt{t_i.move(j)}$ (the train $\mathtt{t_i}$ moves to the segment $\mathtt{r_j}$), $\mathtt{t_i.stop}$ (this action stops the train), $\mathtt{r_i.ggreen}$ (the signal of rail $\mathtt{r_i}$ is set to green) and $\mathtt{r_i.gred}$ (the signal of segment $\mathtt{r_i}$ is set to red).

Recall that in section 3 we introduced a logic with vocabularies that can have several versions of deontic predicates. In this example, we consider three versions of deontic predicates for each train, and one version for each segment. We denote by $\mathtt{t_i}.\mathsf{P}^k()$, $\mathtt{t_i}.\mathsf{P}^k_{\mathsf{w}}()$ and $\mathtt{t_i}.\mathsf{O}^k()$ the permissions and obligations corresponding to train $\mathtt{t_i}$. We use the same notation for the segments. Furthermore, we use some syntactic sugar and instead of writing $\mathtt{t_i}.\mathsf{O}^k(\mathtt{t_i.move(j)})$ we write $\mathtt{t_i}.\mathsf{O}^k(\mathtt{move(j)})$, i.e., we do not repeat twice the trains and the segments when the second occurrence can be deduced from the context.

The axioms are as follows:

$$\textbf{T1.} \quad \bigoplus_{1 \leq j \leq m} \mathtt{t_i.r_j}$$

(for every $1 \leq \mathtt{i} \leq \mathtt{n}$.) ($\bigoplus_{1 \leq j \leq \mathtt{m}} \mathtt{t_i.r_j}$ denotes the exclusive "or" of the predicates $\mathtt{t_i.r_j}$.) This axiom states that each train is in one and only one segment.

$$\textbf{T2.} \quad \neg \mathsf{Done}(\mathbf{U}) \rightarrow \bigwedge_{(1 \leq \mathtt{i} \leq \mathtt{n})} \bigwedge_{(1 \leq \mathtt{j} \leq \mathtt{n}) \wedge (\mathtt{i} \neq \mathtt{j})} \neg(\mathtt{t_i.r_k} \wedge \mathtt{t_j.r_k})$$

This axiom says that, at the beginning of time, there are not two trains in the same segment.

$$\textbf{T3.} \quad \bigvee_{1 \leq \mathtt{i} \leq m} (\mathtt{r_i} R \mathtt{r_j} \wedge \mathtt{t_k.r_i}) \rightarrow \langle \mathtt{t_k.moveto(j)} \rangle \top$$

(for every $1 \leq \mathtt{j} \leq \mathtt{m}$ and $1 \leq \mathtt{k} \leq \mathtt{n}$.) Axioms **T3** say that train $\mathtt{t_k}$ can move to segment $\mathtt{r_j}$ if and only if the train $\mathtt{k}$ is in a segment that is connected to $\mathtt{r_j}$.

$$\textbf{T4.} \quad \neg \mathtt{r_i} R \mathtt{r_i}$$

(for every $\mathtt{i}$.) Segments are not connected with themselves.

$$\textbf{T5.} \quad \Big(\bigvee_{1 \leq \mathtt{i,j} \leq \mathtt{n} \wedge \mathtt{i} \neq \mathtt{j}} \mathtt{t_i.r_k} \wedge \mathtt{t_j.r_k} \Big) \leftrightarrow \mathsf{v_k}$$

(where $1 \leq \mathtt{k} \leq \mathtt{m}$.) There is a violation in segment $\mathtt{r_k}$ if and only if there are two trains in segment $\mathtt{r_k}$.

$$\textbf{T6.} \quad \mathtt{t_i.r_j} \rightarrow \mathtt{r_j}.\mathsf{O}^1(\mathtt{gred})$$

(for every $1 \leq \mathtt{j} \leq \mathtt{m}$ and $1 \leq \mathtt{i} \leq \mathtt{n}$.) When there is a train in a segment, the signal for this segment must be red.

$$\textbf{T7.} \quad (\bigwedge_{1 \leq \mathtt{i} \leq \mathtt{n}} \neg \mathtt{t_i.r_j}) \rightarrow \mathtt{r_j}.\mathsf{O}^1(\mathtt{ggreen})$$

(for every $1 \leq \mathtt{j} \leq \mathtt{m}$.) If there is no train in the segment, then the signal for the segment must be green.

$$\textbf{T8.} \quad \neg \mathtt{t_i.r_k} \wedge \neg \mathtt{r_k.green} \rightarrow \mathtt{t_i}.\mathsf{F}^1(\mathtt{move(k)})$$

(for every $1 \leq \mathtt{i} \leq \mathtt{i}$, $1 \leq \mathtt{k} \leq \mathtt{m}$ and $1 \leq \mathtt{j} \leq 2$.) If the signal of a segment is red, then any train is forbidden to move into the segment.

$$\textbf{T9.} \quad \mathtt{t_i.move(k)} \sqcap \mathtt{t_j.move(k)} =_{act} \emptyset$$

(for every $1 \leq \mathtt{i}, \mathtt{j} \leq \mathtt{n}$ and $\mathtt{i} \neq \mathtt{j}$.) We suppose that two trains cannot enter to the same segment simultaneously.

$$\textbf{T10.} \quad \mathtt{t_j}.\mathsf{F}^1(\mathtt{move(k)}) \rightarrow \mathtt{t_j}.\mathsf{O}^2(\overline{\mathtt{move(k)}} \sqcup \mathtt{stop})$$

(for every $1 \leq \mathtt{j} \leq \mathtt{n}$, $1 \leq \mathtt{k} \leq \mathtt{m}$ and $1 \leq \mathtt{i} \leq 3$.) This axiom formalizes a contrary-to-duty statement: if you are forbidden to move to segment $\mathtt{r_k}$, then you are obliged to not move the train to segment $\mathtt{r_k}$, or to stop the train. This statement also can be read as saying: if you are forbidden to move to segment $\mathtt{r_k}$, and you do it, you have to stop the train. This is similar to the *Gentle Killer* paradox.

Note that we are only taking into account the trains that for some reason ignore a red signal and enter into the segment. We must also specify what happens when another train is already in the segment, to avoid train collisions.

$$\textbf{T11.} \quad \mathtt{t_i.r_j} \wedge \mathtt{r_j.v} \rightarrow \mathtt{t_i}.\mathsf{O}^2(\mathtt{stop})$$

Another bad scenario is when a train is "locked" in a segment, i.e., when a train is in a segment where all the connected segments have their signal set to red. In this case the train is obliged to stop.

$$\textbf{T12.}\quad \mathtt{t_i.r_k} \wedge (\bigwedge_{1 \leq \mathtt{j} \leq \mathtt{m}} (\langle \mathtt{t_i.move(j)} \rangle \top \rightarrow \mathsf{F}^1(\mathtt{t_i.move(j)}))) \rightarrow \mathtt{t_i}.\mathsf{O}^3(\mathtt{stop})$$

The following axiom says that trains cannot move to a segment and at the same time this segment's signal changes to red; we assume some kind of mechanism which prevents a signal from changing at the same moment that a train is entering the segment.

$$\textbf{T13.}\quad \mathtt{t_i.move(j)} \sqcap \mathtt{r_j.gred} =_{act} \emptyset$$

(for every $1 \leq \mathtt{i} \leq \mathtt{n}$ and $1 \leq \mathtt{j} \leq \mathtt{m}$.) We define the behaviour of each action with the following axioms.

$$\textbf{T14.}\quad ([\mathtt{t_i.take(j)}]\mathtt{t_i.r_j}) \wedge (\neg \mathtt{t_i.r_j} \rightarrow [\overline{\mathtt{i.take(j)}}]\neg \mathtt{t_i.stop})$$

$$\textbf{T15.}\quad ([\mathtt{r_j.ggreen}]\mathtt{r_j.g}) \wedge (\neg \mathtt{r_j.g} \rightarrow [\overline{\mathtt{r_j.ggreen}}]\neg \mathtt{r_j.g})$$

$$\textbf{T16.}\quad ([\mathtt{r_j.gred}]\neg \mathtt{r_j.g}) \wedge (\neg \mathtt{r_j.g} \rightarrow [\overline{\mathtt{r_j.gred}}]\neg \mathtt{r_j.g})$$

We can prove some properties of this specification. For example, we can prove that, if obligations of type 2 are fulfilled by trains, then there is no danger of having two trains in the same segment. Let Φ be the following set of formulae:

$$\Phi_1 = \{\mathsf{AG}(\mathtt{t_i}.\mathsf{O}^2(\mathtt{stop}) \rightarrow \mathsf{ANDone}(\mathtt{t_i.stop})) \mid 1 \leq \mathtt{i} \leq \mathtt{n}\}.$$

These (finite) sets of formulae express that trains fulfil the obligations of type 2. We can consider a similar set of formulae for the segments:

$$\Phi_2 = \{\mathsf{AG}(\mathtt{r_i}.\mathsf{O}^1(\mathtt{gred}) \rightarrow \mathsf{ANDone}(\mathtt{r_i.gred})) \mid 1 \leq \mathtt{j} \leq \mathtt{m}\}.$$

Using these sets of formulae, we can prove the following property:

$$\Phi_1, \Phi_2 \vdash_{\textbf{Train}} \neg(\mathtt{t_i.r_k} \wedge \mathtt{t_j.r_k})$$

Informally, when trains fulfil their obligations of stopping at a red signal and segments fulfil the obligation of setting their signal to red when there are trains in the segment, then we cannot have two trains in the same segment.

The proof uses the axiom of induction. Using axiom **T2** and propositional logic we obtain $\vdash_{\textbf{Train}} \neg\mathsf{Done}(\mathbf{U}) \rightarrow \neg(\mathtt{t_i.r_k} \wedge \mathtt{t_j.r_k})$. Now, we prove:

$$\Phi_1, \Phi_2 \vdash_{\textbf{Train}} \neg(\mathtt{t_i.r_k} \wedge \mathtt{t_j.r_k}) \rightarrow [\mathbf{U}]\neg(\mathtt{t_i.r_k} \wedge \mathtt{t_j.r_k}).$$

The proof is a follows:

1.	$\neg \mathtt{t_i.r_k} \wedge \mathtt{t_j.r_k} \rightarrow \mathtt{r_k}.\mathsf{O}^2(\mathtt{gred})$	**T6**
2.	$\mathtt{r_k}.\mathsf{O}^1(\mathtt{gred}) \rightarrow [\mathbf{U}]\mathsf{Done}(\mathtt{r_k.gred})$	PDL, **TempAx1**, Assumption
3.	$\mathtt{r_k.gred} \sqcap \mathtt{t_i.move(k)} =_{act} \emptyset$	**T13**
4.	$\neg \mathtt{t_i.r_k} \wedge \mathtt{t_j.r_k} \rightarrow [\mathtt{t_i.move(k)}]\bot$	PDL, 1,2,3
5.	$\neg \mathtt{t_i.r_k} \rightarrow [\overline{\mathtt{t_i.move(k)}}]\neg \mathtt{t_i.r_k}$	PDL, **T14**
6.	$\neg \mathtt{t_i.r_k} \wedge \mathtt{t_j.r_k} \rightarrow [\mathbf{U}]\neg \mathtt{t_i.r_k}$	PDL, 4, 5
7.	$\mathtt{t_i.r_k} \wedge \neg \mathtt{t_j.r_k} \rightarrow \mathtt{r_k}.\mathsf{O}^1(\mathtt{gred})$	**T6**
8.	$\mathtt{r_k.gred} \sqcap \mathtt{t_j.move(k)} =_{act} \emptyset$	**T13**
9.	$\neg \mathtt{t_j.r_k} \wedge \mathtt{t_i.r_k} \rightarrow [\mathtt{t_j.move(k)}]\bot$	PDL, 1,2,3
10.	$\neg \mathtt{t_j.r_k} \rightarrow [\overline{\mathtt{t_j.move(k)}}]\neg \mathtt{t_i.r_k}$	PDL, **T14**
11.	$\neg \mathtt{t_j.r_k} \wedge \mathtt{t_i.r_k} \rightarrow [\mathbf{U}]\neg \mathtt{t_j.r_k}$	PDL, 4, 5
12.	$\neg \mathtt{t_j.r_k} \wedge \neg \mathtt{t_i.r_k} \rightarrow [\mathtt{t_i.move(k)} \sqcap \mathtt{t_j.move(k)}]\bot$	PDL, **T9**
13.	$\neg \mathtt{t_j.r_k} \wedge \neg \mathtt{t_i.rk} \rightarrow$ $[\overline{\mathtt{t_i.r_k.move(k)} \sqcap \mathtt{t_j.r_k.move(k)}}]\neg \mathtt{t_j.r_k} \wedge \neg \mathtt{t_i.rk}$	PDL, **T14**
14.	$\neg \mathtt{t_j.r_k} \wedge \neg \mathtt{t_i.rk} \rightarrow [\mathbf{U}]\neg \mathtt{t_j.r_k} \wedge \neg \mathtt{t_i.rk}$	PDL, 6, 11, 12, 13
15.	$\neg(\mathtt{t_i.r_k} \wedge \mathtt{t_j.r_k} \rightarrow [\mathbf{U}]\neg(\mathtt{t_i.r_k} \wedge \mathtt{t_j.r_k}))$	PDL, 14, 11, 6

During the proof we use the acronym PDL to point out that basic property of the logic is used (see [6] for a listing of the basic properties). Therefore, using the induction rule, we get $\vdash_{\mathbf{train}} \neg(\mathtt{t_i.r_k} \wedge \mathtt{t_j.r_k})$. Another property is that, when the obligations of type 3 are fulfilled, then when we have two trains in a segment, both will stop. The property can be stated as follows:

$$\mathtt{t_i}.\mathsf{O}^3(\mathtt{stop}) \rightarrow \mathsf{ANDone}(\mathtt{t_i.stop}) \vdash_{\mathbf{Train}} \mathtt{t_i.r_k} \wedge \mathtt{t_j.r_k} \rightarrow \mathsf{AN}\mathtt{t_i.stop} \wedge \mathtt{t_j.stop}.$$

The proof is straightforward from the axioms.

We can think of this property as a recovery property, since from a state where there is a (dangerous) violation we go into a state where we still have a violation but it is safe, since it is free of train collisions. As stated in [4], fault-tolerance is not only about reaching a state free of error after a violation. But also, in some cases, it is acceptable to reach a safe state, where no further violations might arise. Of course, this example can be made more realistic, and we can state that after two trains are stopped in the same segment, then one of them can be removed, or an alternative exit can be made available. We keep the example as simple as

$$\begin{tikzcd}
\underset{\mathtt{t_1.r_1, t_2.r_2, r_1}R\mathtt{r_2}}{s_0} \arrow[r, dashed, bend left, "\mathtt{t_1.move(2)}"] & \underset{\mathtt{t_1.r_2, t_2.r_2, r_2.v, r_1}R\mathtt{r_2, r_2.green}}{s_1} \arrow[loop above, dashed, "\overline{\mathtt{t_1.stop}}"]
\end{tikzcd}$$

Fig. 2. Example of of violation

possible to show how deontic predicates can be used to express requirements over specifications, which, when not fulfilled, yield a violation or bad behaviour.

On the other hand, if obligations of type 2 are not fulfilled, we can reach dangerous states. In figure 2, we have a model with two states s_0 and s_1; below each state, we have the predicates that are true at this state. We have two segments which are connected, and we have two trains, $\mathtt{t_1}$ is in segment $\mathtt{r_1}$ and $\mathtt{t_2}$ is in segment $\mathtt{r_2}$. Since segment $\mathtt{r_2}$ is occupied, $\mathtt{t_1}$ is forbidden to move to that segment, but if it moves, then it must stop. The train moves to that segment and it does not stop. We reach a state where the two trains are in the same segment, and $\mathtt{t_1}$ executes any action but $\mathtt{t_1.stop}$, which will produce a collision in the real world. This model also shows that the contrary to duty predicate expressed by axiom **T10** does not introduce any inconsistency in the specification.

5 Second Example: Byzantine Generals

The Byzantine generals problem was stated originally in [18]; the problem is the following. We have a general with $n-1$ lieutenants. The general and his lieutenants can communicate with each other using messengers. The general may decide to attack an enemy city or to retreat; then, he sends the order to his lieutenants. Some of the lieutenants might be traitors. Traitors might deliver false messages or perhaps they avoid sending a message that they received. The loyal lieutenants must agree on attacking or retreating. This problem is a classic problem of fault-tolerance and distributed computing. Different solutions have been proposed, for example in [18,13,25]. These solutions are simpler when an authenticated way of communication is used, i.e, traitors cannot lie. A solution proposed in the original paper is using signed messages in such a way that signatures cannot be forged (using some encryption protocol). The analogy with fault-tolerance is straightforward: the general is a sender process, the lieutenants are processes that have to agree with some decision taken by the sender. The traitors are faulty processes.

The specification that we provide below uses the ideas introduced in [13,25], where authenticated messages are used. The specification does not assume any form of authentication to prevent forged messages. Instead, deontic predicates are used to express that traitors are forbidden to lie. Of course, they might forge messages anyway. We consider this as a malicious behaviour which is a worse betrayal than to not obey orders. The important point here is that deontic operators allow us to abstract from the mechanisms that are used to prevent traitors from lying.

We have the following actions: $\mathtt{l_i.sendA(j)}$ (lieutenant $\mathtt{l_i}$ sends the message of attack to lieutenant $\mathtt{l_j}$), $\mathtt{l_i.fwd(k, A, j)}$ (lieutenant $\mathtt{l_i}$ forwards to $\mathtt{l_j}$ the message of attack that he received from $\mathtt{l_k}$), $\mathtt{l_i.betray}$ (lieutenant $\mathtt{l_i}$ becomes a traitor). We consider a clock that allows lieutenants to synchronize; the action $\mathtt{tt}$ increments the clock by one unit of time. The specification uses $\mathtt{m}+\mathtt{1}$ rounds of messages, which are coordinated by means of the clock. We have the following predicates: $\mathtt{l_i.A_j}$ (this predicate indicates that $\mathtt{l_i}$ has received a message from $\mathtt{l_j}$ saying that he must attack). We have a violation predicate $\mathtt{l_i.v}$ for each lieutenant (this predicate is true when $\mathtt{l_i}$ is a traitor, i.e., a $\mathtt{l_i}$ is in a violation state) and $\mathtt{l_i.d}$ (this predicate is true when $\mathtt{l_i}$ has decided to attack), $\mathtt{r_i}$ (this predicate is true when we are in round $\mathtt{i}$ od the decision protocol).

We assume that $\mathtt{l_0}$ is the general, the messages are delivered correctly and all the lieutenants can communicate directly with each other, in such a way that they can recognize who is sending a message. We have $\mathtt{n}$ lieutenants and the specification that is shown below uses a constant $\mathtt{m} < \mathtt{n}$ that indicates that the specification tolerates at most $\mathtt{m}$ traitors.

(For the deontic predicates we use the same notation conventions as in the train example.) The axioms are the following. Note that the following are axiom schemas, each formula denotes a finite collection of axioms.

$$\textbf{1.}\quad \neg\mathsf{Done}(\mathbf{U}) \to \Big(\bigwedge_{(1\leq \mathtt{i}\leq n)}\ \bigwedge_{(1\leq \mathtt{j}\leq \mathtt{n})\wedge(\mathtt{i}\neq \mathtt{j})} \neg\mathtt{l_i.A_j}\Big) \wedge \mathtt{r_1} \wedge \Big(\bigwedge_{1\leq \mathtt{i}\leq \mathtt{n}} \neg\mathtt{l_i.d}\Big)$$

At the beginning, the lieutenants have not received any message, we are in the first round and the lieutenants (with the exception of the general) have not taken any decision (by default the decision is to retreat).

$$\textbf{2.}\quad \neg\mathsf{Done}(\mathbf{U}) \wedge \neg\mathtt{l_0.v} \to (\mathtt{l_0.d} \to \mathsf{AG}\mathtt{l_0.d}) \wedge (\neg\mathtt{l_0.d} \to \mathsf{AG}\neg\mathtt{l_0.d})$$

If the general is loyal, then he keeps holding the same decision that he has taken at the beginning.

$$\textbf{3.}\quad \bigwedge_{1\leq \mathtt{i}\leq \mathtt{n}} \mathtt{l_i}.\mathsf{O}^1(\overline{\mathtt{betray}})$$

Lieutenants should not betray.

$$\textbf{4.}\quad \neg\mathsf{Done}(\mathbf{U}) \wedge \mathtt{l_0.d} \to \mathtt{l_0}.\mathsf{O}^2\Big(\bigsqcup_{1\leq \mathtt{i}\leq n} \mathtt{sendA(i)}\Big)$$

At the beginning, if the general decided to attack, then he ought to send a message with his decision to the other lieutenants.

$$\textbf{5.}\quad (\mathtt{r_j} \rightarrow [\mathtt{tt}]\mathtt{r_{j+1}}) \wedge \neg \mathtt{r_m} \rightarrow \langle \mathtt{tt} \rangle \top$$

These axioms specify the behaviour of the clock.

$$\textbf{6.}\quad \mathsf{AN}\mathtt{tt}$$

We always increment the clock.

$$\textbf{7.}\quad \mathtt{r_k} \wedge \mathtt{l_i.A_{j_1}} \wedge \ldots \wedge \mathtt{l_i.A_{j_k}} \rightarrow \mathtt{l_i.d}$$

(where $1 \leq \mathtt{k} \leq \mathtt{m} + 1$, $1 \leq \mathtt{i} \leq \mathtt{n}$ and $1 \leq \mathtt{j_1}, \ldots, \mathtt{j_k} \leq 1$ are $\mathtt{k}$ different numbers.) These axioms indicate that, if in round $\mathtt{k}$ the lieutenant $\mathtt{l_i}$ has received $\mathtt{k}$ messages with the order to attack, then he decides to attack.

$$\begin{aligned}\textbf{8.}\quad & \mathtt{r_k} \wedge \mathtt{l_i.A_{j_1}} \wedge \ldots \wedge \mathtt{l_i.A_{j_k}} \rightarrow \\ & \mathtt{l_i}.\mathsf{O}^2\Big(\big(\bigsqcup_{1 \leq \mathtt{j} \leq n \wedge \mathtt{j} \neq \mathtt{j_1} \ldots \wedge \mathtt{j} \neq \mathtt{j_k}} \mathtt{sendA(j)} \sqcap \mathtt{fwd(j_1, A, j)} \sqcap \ldots \sqcap \mathtt{fwd(j_k, A, j)}\big)\Big)\end{aligned}$$

These axioms indicate that, if in round $\mathtt{r_k}$ the lieutenant $\mathtt{l_i}$ has received $\mathtt{k}$ messages with the order to attack from $\mathtt{k}$ different persons, then he ought to notify all the rest of the lieutenants about the decision to attack; he also forwards all the messages received.

$$\textbf{9.}\quad \mathtt{l_i.v} \wedge \neg \mathtt{l_i.A_j} \rightarrow \mathsf{F}^3\big(\bigsqcup_{1 \leq \mathtt{k} \leq \mathtt{n}} \overline{\mathtt{fwd(j, A, k)}}\big)$$

If a lieutenant is a traitor, then he is forbidden to lie. This involves contrary-to-duty reasoning. Lieutenants might betray at any moment (which is forbidden), but, if they betray, then they must not lie.

$$\mathbf{10.}\quad \mathtt{r_k} \land \neg\mathtt{l_i.v} \land \neg\mathtt{l_i.A_{j_1}} \land \ldots \land \neg\mathtt{l_i.A_{j_t}} \to$$
$$[\bigsqcup_{1 \leq \mathtt{k} \leq \mathtt{n}} \mathtt{sendA(k)}]\bot \land [\bigsqcup_{1 \leq \mathtt{k},\mathtt{k'} \leq \mathtt{n}} \mathtt{fwd(k, A, k')}]\bot$$

(where $1 \leq \mathtt{k} \leq \mathtt{m} + \mathtt{1}$, $1 \leq \mathtt{i} \leq \mathtt{n}$ and $\mathtt{t} > \mathtt{n} - \mathtt{k}$.) These axioms say that, when in round $\mathtt{r_k}$ a loyal lieutenant has not received at least $\mathtt{k}$ messages saying that he must attack, then he does not send nor forward any message.

$$\mathbf{11.}\quad \mathtt{r_{m+1}} \to (\mathtt{l_i.d} \to \mathsf{AG}\mathtt{l_i.d}) \land (\neg\mathtt{l_i.d} \to \mathsf{AG}\neg\mathtt{l_i.d})$$

(for any $1 \leq \mathtt{i} \leq \mathtt{n}$.) These axioms expresses that the decision taken in round $\mathtt{m} + \mathtt{1}$ is final.

$$\mathbf{12.}\quad [\mathtt{l_i.sendA(j)}]\mathtt{l_j.A_i}$$

$$\mathbf{13.}\quad [\mathtt{l_i.fwd(k, A, j)}]\mathtt{l_k.A_j}$$

$$\mathbf{14.}\quad \neg\mathtt{l_i.A_j} \to [\overline{\mathtt{l_j.sendA(i)} \sqcup \bigsqcup_{1 \leq \mathtt{t} \leq \mathtt{n}} \mathtt{l_t.fwd(i, A, j)}}]\neg\mathtt{l_i.A_j}$$

$$\mathbf{15.}\quad \neg\mathtt{l_i.v} \land \mathtt{l_i.d} \to [\mathbf{U}]\mathtt{l_i.d}$$

$$\mathbf{16.}\quad \mathtt{l_i.A_j} \to [\mathbf{U}]\mathtt{l_i.A_j}$$

(for every $1 \leq \mathtt{k}, \mathtt{i}, \mathtt{j} \leq \mathtt{n}$.) These axioms specify the behaviour of the actions $\mathtt{l_i.sendA(j)}$ and $\mathtt{l_t.fwd(i, A, j)}$. Axiom **15** says that, if a loyal lieutenant has decided to attack he sticks with his decision; axiom **16** says that lieutenants do not forget the messages received. Finally, we describe the behaviour of the action `betray`.

$$\mathbf{17.}\quad [\mathtt{l_i.betray}]\mathtt{l_i.v}$$

$$\mathbf{18.}\quad \neg\mathtt{l_i.v} \to [\overline{\mathtt{l_i.betray}}]\neg\mathtt{l_i.v}$$

The axioms of the specification depend on a number $\mathtt{m}$ which, intuitively, is the number of traitors for which the specification ensures that the loyal lieutenants will agree on a decision. We sketch the proof of that fact that, if we have less than $\mathtt{m}$ traitors, then the loyal lieutenants reach an agreement. Consider, first, the following set of formulae:

$$\Phi_1 = \{\mathtt{l_i}.\mathsf{F}^3(\mathtt{fwd(k,A,j)}) \to \mathsf{ANDone}(\overline{\mathtt{l_i.fwd(k,A,j)}}) \mid \text{for any } 1 \leq \mathtt{i,j,k} \leq \mathtt{n}\}$$

This set of formulae say that traitors do not lie. The following formulae say that there are at most $\mathtt{m}$ traitors:

$$\Phi_2 = \mathsf{AG}(\neg \mathtt{l_{j_1}.v} \wedge \ldots \wedge \neg \mathtt{l_{j_{n-m}}.v})$$

(for some different $0 \leq \mathtt{j_1}, \ldots, \mathtt{j_{n-m}} \leq \mathtt{n}$.) Another useful supposition is that loyal lieutenants fulfil their obligations, which is expressed by the following formulae:

$$\Phi_3 = \{\mathtt{l_i}.\mathsf{O}^2(\alpha) \to \mathsf{ANDone}(\alpha) \mid \text{ for every } 1 \leq \mathtt{i} \leq \mathtt{n}\}.$$

Then, if we suppose that there are at least $\mathtt{n} - \mathtt{m}$ lieutenants who are not traitors, traitors do not lie and that loyal lieutenants fulfil their obligations, we can prove that in round $\mathtt{m} + \mathtt{1}$ the loyal lieutenants reach an agreement. This is expressed with the following formulae:

$$\Phi_1, \Phi_2, \Phi_3 \vdash_{\mathbf{Biz_m}} \mathtt{r_{m+1}} \to (\mathtt{l_{j_1}.d} \leftrightarrow \ldots \leftrightarrow \mathtt{l_{j_{n-m}}.d}).$$

(We denote by $\mathbf{Biz_m}$ the specification given above.) This property follows trivially if we prove that any two loyal lieutenants reach an agreement. This is expressed by the following property:

Property 1

$$\Phi_1, \Phi_2, \Phi_3 \vdash_{\mathbf{Biz_m}} \mathtt{r_{m+1}} \to (\mathtt{l_{u_1}.d} \leftrightarrow \mathtt{l_{j_{u_2}}.d}).$$

(for any $\mathtt{u_1}, \mathtt{u_2} \in \{\mathtt{j_1}, \ldots, \mathtt{j_{n-m}}\}$*.)*

Sketch of Proof. At the beginning we have $\neg \mathtt{l_{u_1}.d}$ and $\neg \mathtt{l_{u_2}.d}$. If, in any round $\mathtt{r_k}$ with $\mathtt{k} \leq \mathtt{m}$, we have $\mathtt{l_{u_1}.d}$ by axiom **9**, and since we assume that loyal lieutenants fulfil their obligations, we know that the action $\mathtt{l_{u_1}.sendA(l_{u_2})}$ will be executed and also $\mathtt{l_{u_1}}$ will forward all of the $\mathtt{k}$ messages that he received with an attack order. This implies that, in round $\mathtt{r_{k+1}}$, lieutenant $\mathtt{l_{u_2}}$ will have received $\mathtt{k} + \mathtt{1}$ messages saying attack, and, therefore, by axiom **8**, in round $\mathtt{r_{k+1}}$ we have $\mathtt{l_{u_2}.d}$. The same reasoning can be applied to $\mathtt{l_{u_2}.d}$ in round $\mathtt{r_k}$ with $\mathtt{k} \leq \mathtt{m}$. If $\mathtt{l_{k_1}.d}$ is true in round $\mathtt{r_{m+1}}$ and false in all the earlier rounds, then this lieutenant has received $\mathtt{m} + \mathtt{1}$ messages saying "attack", but since traitors do not lie by assumption and also we assumed that we have at most $\mathtt{m}$ traitors, lieutenant $\mathtt{l_{u_1}}$ have received an order to attack from some loyal lieutenant, which by axiom **9** sent the same orders to lieutenant $\mathtt{l_{u_2}}$; this implies that in the next round after receiving the order from the loyal lieutenant, both have decided to attack by axiom **7**. □

It is interesting to note that when traitors lie, the property shown above is not true. Suppose that we have three lieutenants: $\mathtt{l_0}, \mathtt{l_1}, \mathtt{l_2}$ and $\mathtt{l_1}$ is a traitor. Consider the specification instanced with $\mathtt{m} = 1$ (only one traitor). The model in figure 3 shows a counterexample; we have three states: s_0, s_1, s_2, the initial state is s_0. Below each state the predicates that are true at that state are shown, the predicates which are false are not shown. At the beginning, we have that no lieutenant is a traitor, and that the general $\mathtt{l_0}$ has decided to retreat. Each transition is labelled with the actions that are executed in that transition. In the first transition, $\mathtt{l_1}$

$$\begin{tikzcd} \underset{\mathtt{r}_0}{s_0} \arrow[r, dashed, bend left, "\mathtt{l}_1.\mathtt{betray}\sqcap\mathtt{tt}"] & \underset{\mathtt{r}_1,\mathtt{l}_1.\mathtt{t}}{s_1} \arrow[r, dashed, bend left, "\mathtt{l}_1.\mathtt{fwd(0,A,2)}\sqcap\mathtt{tt}"] & \underset{\mathtt{r}_2,\mathtt{l}_2.\mathtt{d}}{s_2} \end{tikzcd}$$

Fig. 3. Counterexample when traitors lie

becomes a traitor; the dashed arrow indicates that a forbidden action was executed. After that, $\mathtt{l}_1$ lies to $\mathtt{l}_2$ and he forwards a message that he did not receive; this is also a forbidden action. As a consequence, in round $\mathtt{r}_2$, lieutenants $\mathtt{l}_2$ and $\mathtt{l}_0$ do not agree since one has decided to attack and the other to retreat.

6 Conclusions and Further Work

In this paper we have introduced an extension of the deontic action logic presented in [11,10,8,7]. The obtained logic allow us to capture contrary-to-duty statements, which have shown to be hard to deal with by other deontic formalisms. CTD structures are common in fault-tolerance. We have grounded this claim with two examples: first, we described the formalization of a simple train system, and we shown how contrary-to-duty statements arise in this scenario; we prove some properties of the example and we show a scenario where the deontic constraints in the specification are violated, and, as a consequence, a non-desirable system state is reached. As a second example, we have described a specification of the Byzantine Generals problem [18], a classic case study in fault-tolerance. Using this example, we show that deontic predicates allow us to have several layers of reasoning about a specification; this is mainly obtained using stratified norms, where violations of the norms at some level are tolerated, while the violation of norms at other levels are not.

The logic presented in this paper can be extended in several ways to obtain more expressive frameworks. A first extension is to enrich the logic with first-order operators; this will allow us to express interesting properties related to data, and to deal with more complex examples. A second extension is to introduce the notion of module or component. This allows one to simplify designs and to enable compositional reasoning over deontic specifications. We have done this partially in [6], where we have shown, using Dijkstra's example of dining philosophers, how specifications can be modularized and how the reasoning about specifications is simplified by using components. Interestingly, the structuring of specifications enables a modular reasoning about the violations arising during a system execution; undoubtedly this deserves further investigation.

References

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

[2] Jean-Raymond Abrial. Train systems. In *RODIN Book*. Springer, 2006.

[3] Lennart Aqvist. Deontic logic. In D.M.Gabbay and F.Guenther, editors, *Handbook of Philosophical Logic*, volume 2, pages 605–714. Kluwer Academic Publishers, 1984.

[4] Anish Arora. *A Foundation of Fault-Tolerant Computing.* PhD thesis, The University of Texas at Austin, 1992.

[5] P. Blackburn, M.de Rijke, and Y.de Venema. *Modal Logic.* Cambridge Tracts in Theoretical Computer Science 53, 2001.

[6] Pablo F. Castro. *Deontic Action Logics for the Specification and Analysis of Fault-Tolerance.* PhD thesis, McMaster University, Department of Computing and Software, 2009.

[7] Pablo F. Castro and T.S.E. Maibaum. A complete and compact deontic action logic. In *The 4th International Colloquium on Theoretical Aspects of Computing.* Springer Berlin, 2007.

[8] Pablo F. Castro and T.S.E. Maibaum. An ought-to-do deontic logic for reasoning about fault-tolerance: The diarrheic philosophers. In *5th IEEE International Conference on Software Engineering and Formal Methods.* IEEE, 2007.

[9] Pablo F. Castro and T.S.E. Maibaum. Reasoning about system-degradation and fault-recovery with deontic logic. In *Workshop on Methods, Models and Tools for Fault-Tolerance*, 2007.

[10] Pablo F. Castro and T.S.E. Maibaum. A tableaux system for deontic action logic. In *Deontic Logic in Computer Science, 9th International Conference, DEON 2008, Luxembourg, Luxembourg, July 15-18, 2008. Proceedings.* Springer, 2008.

[11] Pablo F. Castro and T.S.E. Maibaum. Deontic action logic, atomic boolean algebra and fault-tolerance. *Accepted for publication in Journal of Applied Logic (Feb 27)*, 2009.

[12] Brian F. Chellas. *Modal Logic: An Introduction.* Cambridge University Press, 1999.

[13] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12:656–666, 1983.

[14] James William Forrester. Gentle murder, or the adverbial samaritan. *The Journal of Philosophy*, 81:193–197, 1984.

[15] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* MIT Press, 2000.

[16] Claude Hennebert and Gérard D. Guiho. SACEM: A fault tolerant system for train speed control. In *The Twenty-Third Annual International Symposium on Fault-Tolerant Computing*, pages 624–628, 1993.

[17] S. Kent, B. Quirk, and T.S.E. Maibaum. Specifying deontic behaviour in modal action logic. Technical report, Forest Research Project, 1991.

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

[19] T. S. E. Maibaum. A logic for the formal requirements specification. Technical report, Imperial College, London.Deliverable R3 for FOREST, 1987.

[20] Paul McNamara. Deontic logic. Technical report, The Stanford Encyclopedia of Philosophy, 2006.

[21] J.J. Meyer. A different approach to deontic logic: Deontic logic viewed as variant of dynamic logic. In *Notre Dame Journal of Formal Logic*, volume 29, 1988.

[22] J.J. Meyer, R.J. Wieringa, and F.P.M. Dignum. The paradoxes of deontic logic revisited: A computer science perspective. Technical Report UU-CS-1994-38, Utrecht University, 1994.

[23] Marek J. Sergot and Robert Craven. The deontic component of action language nC+. *DEON*, pages 222–237, 2006.

[24] Marek J. Sergot and Henry Prakken. Contrary-to-duty obligations. In *DEON 94 (Proc.Second International Workshop on Deontic Logic in Computer Science)*, 1994.

[25] T. K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2:80–94, 1987.