Theoretical Computer Science 455 (2012) 98–122

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Encapsulating deontic and branching time specifications

Pablo F. Castro [a,*], Thomas S.E. Maibaum [b]

[a] Departamento de Computación, FCEFQyN, Universidad Nacional de Río Cuarto and CONICET, Río Cuarto, Córdoba, Argentina
[b] Department of Computing & Software, McMaster University, Hamilton (ON), Canada

ARTICLE INFO

Keywords:
Software specification
Formal methods
Software engineering
Bisimulation
Category theory

ABSTRACT

In this paper, we investigate formal mechanisms to enable designers to decompose specifications (stated in a given logic) into several interacting components in such a way that the composition of these components preserves their encapsulation and internal non-determinism. The preservation of encapsulation (or locality) enables a modular form of reasoning over specifications, while the conservation of the internal non-determinism is important to guarantee that the branching time properties of components are not lost when the entire system is obtained. The basic ideas come from the work of Fiadeiro and Maibaum where notions from category theory are used to structure logical specifications. As the work of Fiadeiro and Maibaum is stated in a linear temporal logic, here we investigate how to extend these notions to a branching time logic, which can be used to reason about systems where non-determinism is present. To illustrate the practical applications of these ideas, we introduce deontic operators in our logic and we show that the modularization of specifications also allows designers to maintain the encapsulation of deontic prescriptions; this is in particular useful to reason about fault-tolerant systems, as we demonstrate with a small example.

1. Introduction

Since the seminal paper of Pnueli [1], temporal logics have become standard formalisms to specify and reason about reactive or non-terminating systems. Diverse versions of temporal logics have been proposed to be applied to different domains; we can divide them into linear temporal logics (e.g., LTL) and branching time logics (e.g., CTL). As stated in [2], the latter kinds of logics are suitable to reason about systems where the course of the computation is non-deterministic (i.e., when we have several possible outcomes in a given scenario); for example, with these kinds of logics it is possible to state that there exists some computation which realizes a desired goal. We assume some familiarity with temporal logics, the interested reader might consult [3] for a comprehensive introduction to this topic.

On the other hand, deontic logics are devoted to the study of the reasoning arising in legal or moral contexts [4], where predicates such as *permitted*, *obliged* and *forbidden* naturally appear. In the past few decades, computer scientists have used deontic logics to reason about computing systems (see [5] for a detailed list of applications of deontic logic in computer science). In particular, we take the ideas presented in [6], where deontic predicates are used to distinguish between the description and the prescription of systems. The description of a system is given in a pre/post-condition style, while the prescriptions are given by means of deontic predicates establishing what the desirable behaviours of the system are. We think that this idea can be useful for fault-tolerance, where violations arise naturally when an abnormal behaviour of the

* Corresponding author. Tel.: +54 0358 4676235.
E-mail addresses: pcastro@dc.exa.unrc.edu.ar (P.F. Castro), tom@maibaum.org (T.S.E. Maibaum).

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.12.016

system occurs, and, therefore, some actions must be executed to recover the system from error states. In the previous work [7] we introduced a simple deontic action logic with temporal operators with the goal of using it to reason about fault-tolerance. In that work, we proposed an axiomatic system for this logic and proved some useful properties of it, such as soundness, completeness, decidability and compactness. Furthermore, we described a case study to illustrate the use of this logic to capture some notions related to fault-tolerance.

In this paper we introduce some modifications to the branching time deontic logic presented in [7] with the aim of obtaining a more general framework where system specifications can be written in a modular way. Towards this goal we use theories to describe components, while a suitable notion of morphisms between components is introduced to express that a component is part of a system (a wider specification). Furthermore, we show that components conserve properties when embedded in a system; from the semantical point of view, conservation of properties implies a relationship of bisimilarity between models, this allows us to guarantee two interesting encapsulation properties of components: firstly, environment actions (i.e., stuttering steps) cannot change the state of a component; and secondly, external or environment actions cannot reduce the internal non-determinism of a component. We show that these encapsulation properties allow us to reason about specifications in a local way, i.e., when proving component properties we can restrict ourselves to the language and axioms of the module being analysed. The ideas are presented using the mentioned deontic logic, but we think that they are general enough to be used with other branching time logics.

We have presented a preliminary version of these ideas in [8]; in this paper we go into the technical details and investigate the mathematical properties of this framework. In addition, as an application of the ideas presented below, we show that the logic introduced in this paper enables us to reason in a modular way about the violations that may occur during the execution of a system. Is our view that the prescriptions (i.e., the deontic formulae which state the desirable behaviour of the system) in specifications must respect the structure of systems, and therefore they must be local to components. For example, in [9] deontic operators are also used in specifications, but the deontic constructs there are used in a "global" way, in the sense that the prescriptions of one component may affect other parts of the system. This is undesirable when performing modular reasoning on specifications, since to prove properties for a module we need to take into account the specifications of other components; increasing the complexity in proofs and making components dependent on other modules, this violates some basic properties of modularization such as information hiding [10]. The logic introduced below maintains the locality of the deontic formulae enabling modular reasoning over prescriptions.

In this paper we mainly follow the philosophy of [11], in the sense that a system is specified by putting together smaller specifications (by means of some categorical constructions [12]). The ideas presented below are also inspired by the logical frameworks presented in [9,13], where Goguen's ideas are applied to a linear temporal logic and, therefore, to specifications of concurrent systems and object oriented systems, respectively.

The paper is organized as follows. In the next section we introduce the basic definitions of the underlying logic, and then we introduce some modifications to this logic to be able to support modular reasoning. We discuss the motivations of the paper in Section 3 and in Section 4 we present a relation of bisimulation which allows us to capture (semantically) the notion of encapsulation discussed above, we also present a suitable proof theory. The notion of *component* is discussed in Section 5. In Sections 6 and 7 we show how the logical machinery described in this paper can be used to reason about faults and we compare this logical framework with related works. Finally, we introduce an example to illustrate the application of these ideas in practice.

2. Preliminaries

In this section we present the syntax and semantics of a simple branching time deontic action logic. The basic definitions are based on those given in [7]. We introduce some modifications to the original definitions to be able to modularize the specifications expressed in this logic.

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*: $a_1, \ldots, a_n$, 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 $\mathcal{V} = \{\mathrm{v}_1, \mathrm{v}_2, \mathrm{v}_3, \ldots\}$ 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 modelled. All these sets are mutually disjoint. Using the primitive actions we define the set Δ of actions as follows:

Definition 2.1. Given a vocabulary $\Delta = \langle \Delta_0, \Phi_0, V_0, I_0 \rangle$, then:

- if $a \in \Delta_0$, then $a \in \Delta$.
- $\emptyset, \mathbf{U} \in \Delta$.
- If α and β are actions, then $\alpha \sqcup \beta, \alpha \sqcap \beta, \overline{\alpha} \in \Delta$.
- No other expression belongs to Δ.

Roughly speaking, $\emptyset$ is the impossible action, $\mathbf{U}$ is the non-deterministic choice of any primitive action. $\alpha \sqcup \beta$ is the non-deterministic choice between α and β. $\alpha \sqcap \beta$ is the parallel execution of α and β, and $\overline{\alpha}$ is the execution of an alternative action to α.

The formulae of this logic (denoted by Φ) are defined as follows.

Definition 2.2. Given a vocabulary $\langle \Delta_0, \Phi_0, V_0, I_0 \rangle$, the set Φ satisfies:

- $\mathsf{B} \in \Phi$.
- If $\varphi \in \Phi_0 \cup V_0$, then $\varphi \in \Phi$.
- If $\varphi, \psi \in \Phi$, then $\varphi \to \psi, \neg\psi \in \Phi$.
- If $S \subseteq \Delta_0$ and $\alpha \in \Delta$, then $\mathsf{Done}_S(\alpha) \in \Phi$.
- If $\varphi \in \Phi$ and $\alpha \in \Delta$, then $[\alpha]\varphi \in \Phi$.
- If $\alpha, \beta \in \Delta$ and $i \in I_0$, then $\mathsf{P}^i(\alpha), \mathsf{P}^i_{\mathsf{w}}(\alpha), \alpha =_{act} \beta \in \Phi$.
- If $\varphi, \psi \in \Phi$, then $\mathsf{AN}\varphi, \mathsf{A}(\varphi\ \mathcal{U}\ \psi), \mathsf{E}(\varphi\ \mathcal{U}\ \psi) \in \Phi$.
- No other expression belongs to Φ.

Some intuition about these formulae is useful. $\psi \to \psi, \neg\psi$ are the standard propositional connectives. $[\alpha]\varphi$ is true, when after executing the action α, φ is true. $\mathsf{P}^i(\alpha)$ is true when the action α is allowed to be executed in any scenario at level i; this deontic operator is called strong permission. $\mathsf{P}^i_{\mathsf{w}}(\alpha)$ is true when the action α is allowed to be executed in some scenarios at level i, this operator is called weak permission. The equation $\alpha =_{act} \beta$ is true when actions α and β produces the same events during their execution. $\mathsf{AN}\varphi, \mathsf{A}(\varphi\ \mathcal{U}\ \psi), \mathsf{E}(\varphi\ \mathcal{U}\ \psi)$ have the standard meaning of branching time logic: $\mathsf{AN}\varphi$ is true when in all paths of execution in the next instant φ is true. $\mathsf{A}(\varphi\ \mathcal{U}\ \psi)$ is true when in every path of execution φ is true until ψ becomes true. $\mathsf{E}(\varphi\ \mathcal{U}\ \psi)$ is true if in some path of execution φ is true until ψ becomes true. Note the constant B, which denotes the beginning of time. The relativized *done* operator ($\mathsf{Done}_S(\alpha)$) is a novel operator, which states that the last action in S executed was α. The particular case: $\mathsf{Done}_{\mathbf{U}}(\alpha)$, says that α was the last action executed. The other standard modal and temporal operators can be defined using the ones introduced above: $\langle\alpha\rangle\varphi \stackrel{\text{def}}{=} \neg[\alpha]\neg\varphi$, $\mathsf{AF}\varphi \stackrel{\text{def}}{=} \mathsf{A}(\top\ \mathcal{U}\ \varphi)$, $\mathsf{EF}\varphi \stackrel{\text{def}}{=} \mathsf{E}(\top\ \mathcal{U}\ \varphi)$, $\mathsf{AG}\varphi \stackrel{\text{def}}{=} \neg\mathsf{EF}\neg\varphi$, $\mathsf{EG} \stackrel{\text{def}}{=} \neg\mathsf{AF}\neg\varphi$ and $\mathsf{EN}\varphi \stackrel{\text{def}}{=} \neg\mathsf{AN}\neg\varphi$, the intuitive reading of these operator is as usual [3]; for example, $\mathsf{AG}\varphi$ has the following intuitive meaning: *for all paths of execution, in every instant φ is true*.

Let us illustrate the differences between the two versions of permission with a small example. Consider the formula $\mathsf{P}(\mathtt{wdraw})$ (where `wdraw` is the action of withdrawing money from a cash machine), it says that it is allowed to withdraw money from the machine. However, it may be convenient to say that the action `wdraw` can only be executed in certain scenarios (e.g., when the machine has enough money), and therefore we have to use a weak permission, i.e.: $\mathsf{P}_{\mathsf{w}}(\mathtt{wdraw})$. We say that an action is forbidden if it is not allowed to be executed in any scenario, hence: $\mathsf{F}^i(\alpha) \stackrel{\text{def}}{=} \neg\mathsf{P}^i_{\mathsf{w}}(\alpha)$. (Note that $\neg\mathsf{P}(\alpha)$ does not capture the correct idea of prohibition, since it is true when some ways of executing α is forbidden.) We say that an action is obliged to occur if it is allowed and any other action is forbidden to be executed, i.e., $\mathsf{O}^i(\alpha) \stackrel{\text{def}}{=} \mathsf{P}^i(\alpha) \wedge \neg\mathsf{P}^i_{\mathsf{w}}(\overline{\alpha})$. The index in the deontic predicates allows us to introduce different levels of normative restrictions. The levels are not necessarily related to each other, but a relation can be added by means of axiomatizations. These levels in the deontic operators allow us to distinguish between the norms of different components, e.g., avoiding that the obligation in a component to execute a given action affects the other components in the system (i.e., these components are not obliged to execute this action). In addition, these stratified levels allow us to deal with contrary-to-duty reasoning; i.e., when obligations arise after a violation of other duties. These kinds of statements are conflictive and many logical paradoxes arise when contrary-to-duty statements are used in traditional deontic logics, see [14] for an introduction.

Let us introduce the semantics of this logic. We follow the ideas of [9], where transitions can be produced by actions in the language or by external components (i.e., this is an *open system* approach in the sense that is given in [15]). (Of course, the notions of encapsulation and locality only make sense in the context of open system semantics; if a component is closed, there is no worry about encapsulation.) Intuitively, each action produces a (finite) set of events during the execution of the system (the events that this action "observes" or "participates in"), and also there are other events produced by actions from other components or from the environment. We define the notion of semantic structure.

Definition 2.3 (*Models*)**.** Given a language $L = \langle \Phi_0, \Delta_0, V_0, I_0 \rangle$, a *L-Structure* is a tuple: $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \{\mathcal{P}^i \mid i \in I_0\}, w_0 \rangle$ where:

- $\mathcal{W}$ is a set of worlds.
- $\mathcal{R}$ 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.
- $\mathcal{E}$ is an infinite, enumerable non-empty set, of (names of) events.
- $\mathcal{I}$ is an interpretation function on predicates and actions; predicates are interpreted as sets of worlds and actions are interpreted as sets of events, that is:
  - 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 $e \in \mathcal{I}(a_1 \sqcup \cdots \sqcup a_n)$: if $e \in \mathcal{I}(\alpha_i) \cap \mathcal{I}(\alpha_j)$, where $\alpha_i \neq \alpha_j \in \Delta_0$, then: $\cap\{\mathcal{I}(\alpha_k) \mid \alpha_k \in \Delta_0 \wedge e \in \mathcal{I}(\alpha_k)\} = \{e\}$.
- w_0 is the initial state.
- $\mathcal{P}^i \subseteq \mathcal{W} \times \mathcal{E}$, for each i is relation stating which events are allowed to be executed in each state.

Roughly speaking, the structure gives us a labelled transition system, whose labels are events, which are produced by some local action(s) or could also correspond to external events. Note that we have a set of events, but actions are only interpreted over finite subsets; in our model each event uniquely denotes the occurrence of a set of actions; otherwise we will have an undesired nondeterminism in our models, as the different ways of executing a primitive action should arise because you can execute it together with other actions (perhaps environmental actions). This property is guaranteed by conditions **I.1** and **I.2**: condition **I.1** states that the isolated execution of an action produces at most a unique event; while, condition **I.2** says that, if we execute all the actions which produce a given event, then the execution of this maximal set of actions produce a unique event. These conditions ensure that every one-point set can be generated from the actions of the component; i.e., the labels in the transitions are uniquely determined by some parallel execution of component (or environmental) actions. In the original formulation of these structures (given in [7]) every event is generated by some action, in this sense the structures presented in this paper are more general. The function $\mathcal{I}$ can be easily extended to interpret any action: $\mathcal{I}(\alpha \sqcap \beta) \stackrel{\text{def}}{=} \mathcal{I}(\alpha) \cap \mathcal{I}(\beta)$, $\mathcal{I}(\alpha \sqcup \beta) \stackrel{\text{def}}{=} \mathcal{I}(\alpha) \cup \mathcal{I}(\beta)$ and $\mathcal{I}(\overline{\alpha}) \stackrel{\text{def}}{=} \bigcup\{\mathcal{I}(a_i) \mid a_i \in \Delta_0\} \cap \overline{\mathcal{I}(\alpha)}$. Note that action complement is interpreted as a relative set complement.

We call *standard models* those structures where $\mathcal{E} = \bigcup_{\alpha \in \Delta_0} \mathcal{I}(\alpha)$, i.e., when we do not have "outside" events in the structure. Note that the semantics of the logic described in [7] is given only in terms of standard models. We use maximal traces to give the semantics of the temporal operators. Given a L-structure $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \{\mathcal{P}^i \mid i \in I_0\}, w_0 \rangle$, an infinite trace (or path) is a sequence $\pi = w_0 \stackrel{e_0}{\to} w_1 \stackrel{e_1}{\to} w_2 \stackrel{e_2}{\to} \cdots$ (where each $w_i \stackrel{e_i}{\to} w_{i+1}$ is a labelled transition in M); we denote by $\pi^i = w_i \stackrel{e_i}{\to} w_{i+1} \stackrel{e_{i+1}}{\to} \cdots$ the subpath of π starting at position i. The notation $\pi_i = w_i$ is used to denote the i-th state in the path, and we write $\pi[i, j]$ (where $i \leq j$) for the subpath $w_i \stackrel{e_i}{\to} \cdots \stackrel{e_j}{\to} w_{j+1}$. $\pi(i)$ denotes the event e_i. Finally, given a finite path $\pi' = w'_0 \stackrel{e'_0}{\to} \cdots \stackrel{e'_n}{\to} w_{n+1}$, we say $\pi' \preceq \pi$ if π' is an initial subpath of π, that is: $w_i = w'_i$ and $e_i = e'_i$ for $0 \leq i \leq n$, and we denote by $\prec$ the strict version of $\preceq$. We denote by $\#\pi$ the length of the trace π; if it is infinite we say $\#\pi = \infty$. The set of sequences in M starting in w_0 is denoted by $\Sigma(w_0)$, and the set of maximal sequences starting at w_0 is denoted by $\Sigma^*(w_0)$.

Since, in a trace, we have events that do not belong to the actual component, we need to distinguish between those events generated by the component being specified and those which are from the environment. Given language $L = \langle \Delta_0, \Phi_0 \rangle$, a L-structure M and a maximal path π in M, we define the set:

$$\mathrm{Loc}_{\{a_1,\ldots,a_m\}}(\pi) = \{i \mid i > 0 \wedge \pi(i-1) \in \mathcal{I}(a_1 \sqcup \cdots \sqcup a_m)\}$$

where $\{a_1, \ldots, a_m\} \subseteq \Delta_0$. That is, this set contains all the positions of π where events occur that are observed by some action in $\{a_1, \ldots, a_m\}$. Furthermore, we define:

$$\mathrm{Loc}_L(\pi) = \mathrm{Loc}_{\Delta_0} \cup \{0\}.$$

Roughly speaking, this set contains all the positions of π where events occur that are observed by some action in L. This set is totally ordered by the usual relationship $\leq$. These sets are useful when we want to reason about a restricted part of a system, as we show later on. In the following, given a set S of naturals, we denote by $\min_p(S)$ the minimum element in S which satisfies the predicate p, and similarly for $\max_p(S)$; if S is empty or p is false, then these expressions are undefined. Using these concepts, we define the relationship $\models_L$ between structures and formulae of a given language L (we omit the L when it is understood by context). Note that we introduce the definition of the semantics in a similar way to the presentation in [7], but taking into account the separation between local and external events.

Definition 2.4. Given a trace $\pi = w_0 \stackrel{e_0}{\to} w_1 \stackrel{e_1}{\to} w_2 \stackrel{e_2}{\to} \cdots \in \Sigma^*(w_0)$, we define the relation $\models_L$ as follows:

- $\pi, i, M \models_L \mathsf{B} \stackrel{\text{def}}{\Longleftrightarrow} i = 0$.
- If $p_j \in \Phi_0 \cup V_0$, then $\pi, i, M \models_L p_j \stackrel{\text{def}}{\Longleftrightarrow} \pi_i \in \mathcal{I}(p_j)$.
- $\pi, i, M \models_L \alpha =_{act} \beta \stackrel{\text{def}}{\Longleftrightarrow} \mathcal{I}(\alpha) = \mathcal{I}(\beta)$.
- $\pi, i, M \models_L \mathsf{P}^j(\alpha) \stackrel{\text{def}}{\Longleftrightarrow} \forall e \in \mathcal{I}(\alpha) : \mathcal{P}^j(\pi_i, e)$.
- $\pi, i, M \models_L \mathsf{P}^j_{\mathsf{w}}(\alpha) \stackrel{\text{def}}{\Longleftrightarrow} \exists e \in \mathcal{I}(\alpha) : \mathcal{P}^j(\pi_i, e)$.
- $\pi, i, M \models_L \neg\varphi \stackrel{\text{def}}{\Longleftrightarrow} \text{not } \pi, i, M \models_L \varphi$.
- $\pi, i, M \models_L \varphi_1 \to \varphi_2 \stackrel{\text{def}}{\Longleftrightarrow}$ either not $\pi, i, M \models_L \varphi_1$ or $\pi, i, M \models_L \varphi_2$, or both.
- $\pi, i, M \models_L \mathsf{Done}_S(\alpha) \stackrel{\text{def}}{\Longleftrightarrow} \exists j : j = \max_{<i}(\mathrm{Loc}_S(\pi)) \wedge e_j \in \mathcal{I}(\alpha)$.
- $\pi, i, M \models_L [\alpha]\varphi \stackrel{\text{def}}{\Longleftrightarrow} \forall \pi' = s'_0 \stackrel{e'_0}{\to} s'_1 \stackrel{e'_1}{\to} \cdots \in \Sigma^*(w)$ such that $\pi[0, i] \prec \pi'$, if there is a j such that $j = \min_{>i}(\mathrm{Loc}(\pi'))$, and if $e'_j \in \mathcal{I}(\alpha)$, then $\pi', j, M \models_L \varphi$.
- $\pi, i, M \models_L \mathsf{AN}\varphi \stackrel{\text{def}}{\Longleftrightarrow}$ if $i = \#\pi$, then $\pi, i, M \models \varphi$. If $i \neq \#\pi$, then $\forall \pi' \in \Sigma^*(w)$; $\pi[0..i] \prec \pi'$:, if there is a j such that $j = \min_{>i}(\mathrm{Loc}(\pi'))$, then $\pi', j, M \models \varphi$.

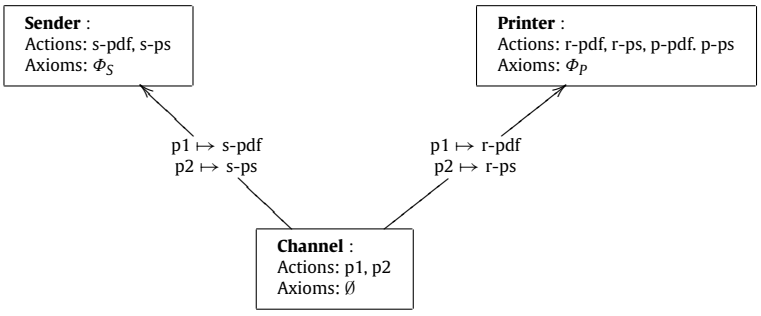


Fig. 1. Example of a Sender–Printer system.

- $\pi, i, M \models_L \mathsf{A}(\varphi_1 \ \mathcal{U} \ \varphi_2) \stackrel{\text{def}}{\Longleftrightarrow}$ if $i = \#\pi$, then $\pi, i, M \models \varphi_2$. If $i \neq \#\pi$, then $\forall \pi' \in \Sigma^*(w) : \pi[0..i] \prec \pi'$ we have that $\exists j \in \text{Loc}((\pi')^i) : \pi', j, M \models \varphi_2$ and $\forall i \leq k \leq j : k \in \text{Loc}((\pi')^i)$, then $\pi', k, M \models \varphi_1$.
- $\pi, i, M \models_L \mathsf{E}(\varphi_1 \ \mathcal{U} \ \varphi_2) \stackrel{\text{def}}{\Longleftrightarrow}$ if $i = \#\pi$, then $\pi, i, M \models \varphi_2$. If $i \neq \#\pi$, then $\exists \pi' \in \Sigma^*(w) : \pi[0..i] \prec \pi'$ such that $\exists j \in \text{Loc}((\pi')^i) : \pi', j, M \models \varphi_2$ and $\forall i \leq k \leq j : k \in \text{Loc}((\pi')^i)$, then $\pi', k, M \models \varphi_1$.

We say that $M \models_L \varphi$ when $\pi, i, M \models_L \varphi$ for every path π and instant i. And we say that $\models_L \varphi$ when $M \models_L \varphi$ for every model M. Note that we use the set $Loc(\dots)$ to observe the events that are only produced by local actions. We can think of the propositional variables in L as local variables, which cannot be changed by other components, i.e., we must require (as is done in [9,13]) that external events do not produce changes in local variables. In [13] the notion of a *locus* trace is introduced to reflect this property in the logic; a locus (trace) is one in which the external events do not affect the state of local variables. However, the logic used in that work is a linear temporal logic, and this implies that here we cannot restrict only to traces to express this requirement, since we have a branching temporal logic. In the following we take further the ideas introduced in [13] and we define *locus models* which have the property of generating locus traces.

The logic introduced in this section can be classified as a variation of boolean modal logic (BML) [16]. However, there are some differences. First, boolean modal logic uses a relational semantics following the tradition of multimodal logics [17]. The boolean complement used in these logics is with respect to the universal relation; this has as a consequence that the window modal operator becomes definable. Similarly, it is possible to define the universal relationship, which allows us to state that a given property is true in every state. BML is NExpTime complete, but when restricted to a finite number of relations it is ExpTime complete [18]. We do not take the relational approach to define the semantics of our logic, instead we use an algebra of events to interpret the actions. This allows us to obtain a straightforward semantics of boolean operators with a relative universal action. In particular, the algebraic approach allows us to obtain a canonical model using the atoms of the Lindenbaum–Tarski boolean algebra of actions. Note that the atoms in our boolean action algebra denote basic transitions of the system being specified. Moreover, these atoms allow us to capture the principle that any basic step is allowed or forbidden. Another difference is that we only provide a finite number of actions, although this can also be done in modal boolean logics it is not straightforward to obtain the canonical model with the relational approach. In [19] we have provided a tableaux system which is in PSPACE; the improvement with respect to modal boolean logics comes from the fact that the complement is not a universal one in our logic.

Broersen [20] defines a similar deontic action logic; however, Broersen's approach uses violation markers to define permissions and prohibitions, and so this implies some differences between the properties of our logic and Broersen's formalism, see [21] for a detailed comparison between these logics. The $\mathsf{Done}(-)$ operator has been discussed in [22–24], here we have introduced a generalized version of this operator to be able to reason about composition of components. This operator can be defined augmenting vocabularies and adding extra-axioms (see [21]); for the sake of simplicity we have included it in the definition of the logic.

3. Components, locality and models

In Section 2 we defined an *open semantics* [15] for our temporal logic, i.e., we consider that components are embedded in a wider system, where we have some state transitions in which the component being defined does not participate. These are transitions performed by other components of the system. Taking this approach allows us to compose logical specifications, as it is shown in [15,13]. Let us illustrate the basic ideas of this paper by a simple example. Consider a component **Printer** that specifies the behaviour of a printer, and a component **Sender** that describes the behaviour of a process that sends documents to the printer. The components communicate each other by means of two ports: one port is used to send *pdf* documents, and the other one to print *ps* documents. The design of this simple system is illustrated in Fig. 1. We describe the sets Φ_S, Φ_P of axioms below. First, let us note the way in which components are connected, we have a component **Channel** which provides two ports, this component has no axioms; the arrows between components identify logical morphisms (functions mapping languages to languages and axioms to axioms, in a coherent way). These mappings allow us to coordinate symbols

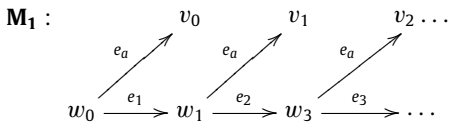


Fig. 2. Example of a non-deterministic execution of a system with executions of non-local events.

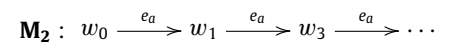


Fig. 3. Example of a non-deterministic execution of a system with only local executions.

of components **Sender** and **Printer**; in this case, these mappings state that the action s-pdf coordinates with the action r-pdf, and similarly for s-ps and r-ps; using the colimit construction we can build a new theory (the system specification); the language of this theory is obtained renaming the symbols of the components in such a way that symbols connected by morphisms (e.g., s-pdf and s-ps) are renamed to the same symbols, and the other symbols are renamed in such a way that symbol clashes are avoided; therefore, the set of axioms of the system is the union of the axioms of the components, using the symbol renaming described above. These ideas are introduced in [13] for specifying concurrent modular systems using logical theories expressed in linear temporal logic; in this setting, morphisms between logical theories capture the notion *being-part-of* which allows specifiers to put together different components; for instance, in the example above the component **Channel** is part of components **Sender** and **Printer**, in the same way these two components are part of the complete system (the colimit). The semantic structures considered in [13] are linear executions of systems, where each instant is "observed"[1] by a set of actions: the actions that are executed in that instant. In this setting, *encapsulation* or *locality* is the property that only local actions may modify a component state. In [13] the following axiom, called the locality axiom, is introduced to capture the notion of encapsulation:

$$\left(\bigwedge_{g\in\Gamma} \exists x_g : g(x_g)\right) \vee \left(\bigwedge_{a\in A} \forall x_a : (\mathsf{X}(a(x_a)) = a(x_a))\right).$$

We explain briefly this axiom here. (For the details the reader can consult [13].) The axiom says that either, an action of the component is executed, or the local data suffer no change. In this formula, Γ is the set of actions of the component, and A is the set of attributes of the actual component and X is the next operator of Fiadeiro–Maibaum's logic. Note that the property of locality can be expressed in one formula since the number of actions of a component is finite. Barringer [15] uses the last component of this formula to capture silent transitions, i.e., transitions that do not affect the state of the component. In temporal linear logic this axiom expresses correctly the notion of encapsulation. However, when we consider scenarios where the time is non-linear, this formula is not expressive enough to capture the notion of encapsulation. The main problem arises when the execution of external actions affects a component's behaviour. Let us illustrate this with some examples. In Fig. 2 an example of a non-deterministic execution of a given system is shown. In this example we assume that the local data is preserved in every transition, the e_i labels denote the occurrence of external events, and the label e_a denotes the occurrence of a local action named a. Suppose that the axiom $\vdash \langle a\rangle\top$ belongs to the component. In this case, it might be the case that the external events: $e_0, e_1, e_2, \ldots$ are executed, which implies that the actual component diverges by means of an infinite execution of external events. This behaviour is not possible when the component is considered in isolation, as it is shown in Fig. 3. One of the principles (inherent to encapsulation) that we want to preserve when reasoning about components is the following induction rule: $\{\mathsf{B} \to \varphi, \varphi \to [\mathbf{U}]\varphi\} \vdash \mathsf{AG}\varphi$; that is, if a property holds when the component starts, and any component action preserve this property, then any component execution satisfies the property. This induction rule is not valid in the example presented above. For instance, the property AGAN*done*(a) is not true in model $\mathbf{M_1}$. This scenario, in particular, may happen when a unfair scheduler is used. Let us take another look at the Printer–Sender example. Suppose that action s-pdf produces the event e_{pdf} and action s-ps produces event e_{ps}. We consider the following set of axioms (i.e., the set Φ_S) in the **Sender** component:

- $\vdash$ [s-pdf $\sqcap$ s-ps]$\bot$,
- $\vdash \mathsf{B} \to \langle$s-ps$\rangle\top \wedge \langle$s-pdf$\rangle\top$,
- [s-ps]($\langle$s-ps$\rangle\top \wedge \langle$s-pdf$\rangle\top$),
- [s-pdf]($\langle$s-ps$\rangle\top \wedge \langle$s-pdf$\rangle\top$).

The first axiom says that actions s-ps and s-ps cannot be executed in parallel; the second formula says that, at the beginning of time, the sender could send a *ps* or a *pdf* document; the other formulae are similar, they say that after sending a *ps* or *pdf* document, the sender is able to send again a *ps* or *pdf* one. In a closed system, a possible model of this theory is $\mathbf{M}'_2$

[1] We follow the terminology introduced by Fiadeiro and Maibaum.

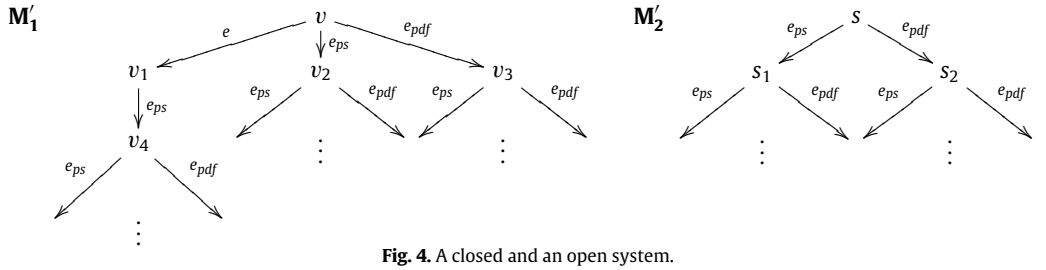


Fig. 4. A closed and an open system.

depicted in Fig. 4; in this structure we have an infinite number of states. We show the first three states: s, s_1, s_2; from s we can execute s-ps and get s_1, or execute s-pdf and reach s_2, and then the pattern is repeated. Consider now an open semantics; this means that actions from the environment (i.e., the printer in this case) may be executed at any time. Then, the structure $\mathbf{M'_1}$ is a possible interpretation of this component/theory; this structure is similar to $\mathbf{M'_2}$ but in state v a non-local action can be executed, and therefore we reach the state v_1 (the event e denotes the occurrence of an external action). As we mentioned above, environment steps are intended to be silent to the component. However, note that in this case, after executing the environment action, we lose the possibility of executing s-pdf. Note that the two structures satisfy the axioms; however, we have $\mathbf{M'_2} \models \langle\text{s-ps}\rangle\top \wedge \langle\text{s-pdf}\rangle\top$ and $\mathbf{M'_1} \not\models \langle\text{s-ps}\rangle\top \wedge \langle\text{s-pdf}\rangle\top$. This property is a consequence of the induction rule presented above, i.e., under the presence of external actions the induction rule is no longer valid. Summarizing, in addition to asking for non-local actions to preserve the component state, we also must require that they preserve the branching arising from the internal non-determinism of the component. In the following sections we will develop the formal machinery needed to characterize the open models that preserve the internal non-determinism of a component, and therefore this will enable, for example, the use of the induction rule.

4. Bisimulation and locus models

In this section we present a notion of bisimulation that allows us to capture the concept of *locus model*: a locus model is a possible behaviour of a component where non-local transitions are present and they behave as identity steps, not only preserving a state configuration, but also preserving the branching arising during the execution of a component.

Given a structure M over a language L, we say that an event e is *non-local* if it does not belong to the interpretation of any action of the language; otherwise, we say that it is a *local event*. We say $w \stackrel{\epsilon}{\Rightarrow} w'$, if there exists a path $w \stackrel{e_0}{\rightarrow} w_1 \stackrel{e_1}{\rightarrow} w_2 \stackrel{e_2}{\rightarrow} \cdots \stackrel{e_n}{\rightarrow} w_n$ in M, such that e_i is non-local for every $0 \leq i \leq n$. We say that $w \stackrel{\infty}{\Rightarrow}$ when there is an infinite path from w: $w \stackrel{e_0}{\rightarrow} w_1 \stackrel{e_1}{\rightarrow} \cdots$, such that every e_i is non-local. Furthermore, we say $w \stackrel{e}{\Rightarrow} w'$ (where e is local) if $w \stackrel{\epsilon}{\Rightarrow} w''$ and $w'' \stackrel{e}{\rightarrow} w'$.

Given two L-structures $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \{\mathcal{P}^i \mid i \in I_0\}, w_0\rangle$ and $M' = \langle \mathcal{W}', \mathcal{R}', \mathcal{E}', \mathcal{I}', \{\mathcal{P}'^i \mid i \in I_0\}, w'_0\rangle$, such that $\mathcal{I}(\alpha) = \mathcal{I}'(\alpha)$ for any α, we say that a relationship $Z \subseteq \mathcal{W} \times \mathcal{W}'$ is a *local bisimulation* between M and M' iff:

- If wZv, then $L(w) = L(v)$.
- If wZv, and $w \stackrel{\infty}{\Rightarrow}$, then either $v \stackrel{\infty}{\Rightarrow}$ or there is a v' such that $v \stackrel{\epsilon}{\Rightarrow} v'$ and v' has no successors by $\rightarrow$ in M'.
- if wZv and $w \stackrel{e}{\rightarrow} w'$. then $w'Zv$ if e is non-local. Otherwise we have some v' such that $v \stackrel{e}{\rightarrow} v'$ and $w'Zv'$.
- $Z^\smile$ also satisfies the above conditions (where $Z^\smile$ is the converse of Z).

Here $L(v)$ denotes the set of all the state formulae (primitive propositions, deontic predicates and equations) true at state v. In branching bisimulation (as defined in [25]), we can "jump" through non-local events; however, here we require a stronger condition: we can move through non-local events, but, if we have the possibility of executing a local event, we must have the same possibility in the related state. We see later on that this notion of bisimulation induces useful properties on the models and that we can characterize this notion in an axiomatic way.

We say that two models M and M' are (locally) *bisimilar* iff $w_0 Z w'_0$ (where w_0 and w'_0 are the corresponding initial states) for some local bisimulation Z; we denote this situation by $M \sim_Z M'$. We prove later on that two bisimilar models are indistinguishable by our logic. In [25], it is shown that *CTL**-*X* (*CTL** without the next operator) cannot distinguish between Kripke structures which are DSS (*divergent sensitive stuttering*) bisimilar; however, in the semantics of the temporal logic considered in that work, there are no labels on the transitions and, therefore, the next operator is problematic since it is interpreted as a global next operator. Here we can take advantage of the fact that we have the events as labels of transitions, and, therefore, we can distinguish between local and non-local transitions. Furthermore, note that our next operator is a local one (although this implies some subtle technical points when it comes to defining the composition of components, see below). Lamport [26,27] rules out the next operator since it is problematic when working on hierarchical decomposition and refinement of temporal specifications, in particular, when stuttering is present; however, as argued in [28] the next operator arises naturally when reasoning about the consequences of actions (e.g., TLA uses primed variables to talk about the next state of program variables); moreover, the next operator is important to state the induction rule and for the recursive characterization of other temporal operators (e.g., **TempAx4** and **TempAx4** below); notice that this logical operator can be captured using modalities and the universal action (**TempAx1** and **TempAx2** below). Note that in this paper we do not deal

with refinement or vertical structuring, we leave this as future work. Below we present the main results about bisimulations; first, let us extend the definition of bisimulation to paths.

Definition 4.1. Given a path $\pi = w_0 \xrightarrow{e_0} w_1 \xrightarrow{e_1} \cdots$ in M and a path $\pi' = v_0 \xrightarrow{d_0} v_1 \xrightarrow{d_1} \cdots$ in M', and a local bisimulation between M and M' such that $M \sim_Z M'$, we say that $\pi Z \pi'$, iff when $w_i Z v_j$, then:

- If we have $w_i \xrightarrow{e_1} \cdots \xrightarrow{e_n} w_n$ in π, with e_j non-local for $1 \leq j \leq n$, then we have $v_j \xrightarrow{d_1} \cdots \xrightarrow{d_m} v_m$ in π', with d_l non-local for every $1 \leq l \leq m$, such that $w_n Z v_m$.
- If we have $w_i \xrightarrow{e} w_{i+1}$ in π, where e is a local action, then we have a (sub)path in π': $v_j \xrightarrow{d_1} \cdots \xrightarrow{e} v_m$ such that for all $1 \leq l < m$, d_l are non-local, and $w_n Z v_m$.
- We also have $\pi' Z^{\smile} \pi$.

This is similar to the definition of stuttering equivalence, but taking into account the labels. Our first property about paths and local bisimulation says that bisimilar initial segments of paths can be extended to bisimilar full paths:

Theorem 4.1. *If $\pi[0..i] Z \pi'[0..j]$, then there exists a $\pi'[0..j] \preceq \pi_2$ such that $\pi Z \pi_2$.*

It is worth noting that, since $Z^{\smile}$ satisfies the same conditions as Z, we have that the above theorem also is true when we replace Z by $Z^{\smile}$. Note that, if $\pi Z \pi'$, we can define a mapping f_π between positions of π and positions of π' as follows, $f_\pi(0) = 0$ and:

$$f_\pi(n+1) = \begin{cases} f_\pi(n) & \text{if } e_n \text{ is non-local} \\ \min_{>f_\pi(n)}(\mathrm{Loc}(\pi')) & \text{otherwise} \end{cases}$$

where $\pi = w_0 \xrightarrow{e_0} w_1 \xrightarrow{e_1} \cdots$. In the same way we can define a function $f_{\pi'}$. Using these functions, we can prove that there exists a tight relationship between the positions of two bisimilar paths.

Property 4.1. *If $\pi Z \pi'$, then $\pi_i Z \pi'_{f_\pi(i)}$, for every position i of π.*

Property 4.2. *If $\pi Z \pi'$, $\#Loc(\pi[0..i]) = \#Loc(\pi'[0..f_\pi(i)])$.*

Corollary 4.1. *If $\pi Z \pi'$, then either:*

- $\pi_i = \pi_{f_{\pi'}(f_\pi(i))}$, *or*
- $\pi_i \overset{\epsilon}{\Rightarrow} \pi_{f_{\pi'}(f_\pi(i))}$, *or*
- $\pi_{f_{\pi'}(f_\pi(i))} \overset{\epsilon}{\Rightarrow} \pi_i$

Using these properties, we obtain the following result which resembles the properties of Galois connections.

Property 4.3. *If $\pi Z \pi'$, then: $\pi'_{f_\pi(i)} \overset{e}{\Rightarrow} \pi'_k$ in π' iff $\pi_i \overset{e}{\Rightarrow} \pi_{f_{\pi'}(k)}$.*

Our first important theorem says that bisimilar (full) paths satisfy the same properties:

Theorem 4.2. *If $\pi Z \pi'$, then, for all positions i, $\pi, i, M \vDash \varphi \Leftrightarrow \pi', f_\pi(i), M \vDash \varphi$.*

As a corollary, we get that local bisimilar structures satisfy the same predicates.

Theorem 4.3. *If $M \sim_Z M'$, then $M \vDash \varphi$ iff $M' \vDash \varphi$.*

Proof. Suppose that $M \vDash \varphi$ and $M' \nvDash \varphi$; therefore, we have that $\pi, i, M' \nvDash \varphi$ for some full path π and position i. But then we get by the theorem above that $\pi', f_{\pi'}(i), M \nvDash \varphi$ for some $\pi' Z \pi$ (which exists since $M \sim_Z M'$). The other direction is similar. □

Let us use this notion of bisimulation to formalize the idea of *locus structure* that, as shown later on, will be essential in defining composition of modules (or components). Roughly speaking, *locus models* are those which have a behaviour which is, essentially, the same as that of a standard model. Hence, the usual notion of encapsulation, as informally understood in software engineering, applies to our concept of component: only local actions can modify the values of local variables, and hence to affect the internal non-determinism of a component. A *locus* trace is one in which, after executing a non-local event, the local variables retain their value. Furthermore, since we have a branching time logic and a modal logic, here it is not enough to just put restrictions on traces. We need to take into account the branching occurring in the semantic structures. Roughly speaking, locus models are those which are locally bisimilar to a standard model. In some sense, this definition characterizes those models which behave as standard models, where the external actions are silent with respect to local attributes and preserve internal non-determinism.

Definition 4.2. Given a language L, we say that a L-structure M' is a locus iff there is a standard model M such that $M \sim_Z M'$ for some local bisimulation Z.

Using the result presented above about local bisimulation, we get that locus structures do not add anything new to the logic (w.r.t. formula validity):

Theorem 4.4. *If M is a locus structure, then $M \models \varphi$ iff there is some standard structure M' such that $M' \models \varphi$.*

Summarizing, nothing is gained or lost in using the locus models of a given language. However, we want to use these kinds of models over a wider notion of logical system; we shall consider several languages and translations between them, and therefore we need to have a notion of model which agrees with the locality properties of a language when we embed this language in another. First, let us define what a translation between two languages is.

Definition 4.3. A translation τ between two languages $L = \langle \Delta_0, \Phi_0, V_0, I_0 \rangle$ and $L' = \langle \Delta'_0, \Phi'_0, V'_0, I'_0 \rangle$ is given by:

1. A mapping $f : \Delta_0 \to \Delta'_0$ between the actions of L and the actions of L',
2. A mapping $g : \Phi_0 \to \Phi'_0$ between the propositions of L and the predicates of L',
3. A mapping $h : V_0 \to V'_0$, between the violations of L and the violations of L',
4. A mapping $i : I_0 \to I'_0$.

That is, a translation describes a mapping between two languages, in such a way that one of them is embedded in the other one. The collection of all the languages and all the translations between them forms the category **Sign**. It is straightforward to see that it is really a category: identity functions define identity arrows, and composition of functions gives us the composition of translations. Now, given a translation, we can extend this translation to formulae (actually we can describe a grammar functor which reflects these facts, as done in Institutions [29] or π-Institutions [30]). Given a translation $\tau : L \to L'$ as explained above, we extend τ to a mapping between the formulae of L and those of L', as follows. For action terms we define:

- $\tau(\alpha \sqcup \beta) \stackrel{\text{def}}{=} \tau(\alpha) \sqcup \tau(\beta)$,
- $\tau(\emptyset) \stackrel{\text{def}}{=} \emptyset$,
- $\tau(\alpha \sqcap \beta) \stackrel{\text{def}}{=} \tau(\alpha) \sqcap \tau(\beta)$,
- $\tau(\overline{\alpha}) \stackrel{\text{def}}{=} \tau(\mathbf{U}) \sqcap \overline{\tau(\alpha)}$,
- $\tau(\mathbf{U}) \stackrel{\text{def}}{=} \tau(a_1) \sqcup \cdots \sqcup \tau(a_n)$ (where $\Delta_0 = \{a_1, \ldots, a_n\}$).

Note that the complement is translated as a relative complement, and the universal action is translated as the non-deterministic choice of all the actions of the original component (which is different from the universal action in the target language). It is important to stress that some extra axioms must be added to the axiomatic system to deal with the fact that the actions are interpreted as being relative to a certain universe. The extension to formulae is as follows:

- $\tau([\alpha]\varphi) \stackrel{\text{def}}{=} [\tau(\alpha)]\tau(\varphi)$, $\tau(\neg\varphi) \stackrel{\text{def}}{=} \neg(\tau(\varphi))$, $\tau(\varphi \to \psi) \stackrel{\text{def}}{=} \tau(\varphi) \to \tau(\psi)$, $\tau(\mathsf{B}) \stackrel{\text{def}}{=} \mathsf{B}$
- $\tau(\mathsf{AN}\varphi) \stackrel{\text{def}}{=} (\langle \tau(\mathbf{U}) \rangle \top \to \mathsf{AN}(\mathsf{Done}(\tau(\mathbf{U}))) \to \tau(\varphi))) \vee ([\tau(\mathbf{U})]\bot \to \tau(\varphi))$
- $\tau(\mathsf{A}(\varphi\ \mathcal{U}\ \psi)) \stackrel{\text{def}}{=} \mathsf{A}(\tau(\varphi)\ \mathcal{U}\ \tau(\psi))$
- $\tau(\mathsf{E}(\varphi\ \mathcal{U}\ \psi)) \stackrel{\text{def}}{=} \mathsf{E}(\tau(\varphi)\ \mathcal{U}\ \tau(\psi))$
- $\tau(\mathsf{Done}_S(\alpha)) \stackrel{\text{def}}{=} \mathsf{Done}_{\tau(S)}(\tau(\alpha))$, where $\tau(S) = \{\tau(a_i) \mid a_i \in S\}$

Note that the formula $\mathsf{AN}\varphi$ is translated to a formula that is true when, if we restrict ourselves to the actions of the original component, the formula $\tau(\varphi)$ is true in the next state. The other cases are obtained by preserving the logical connectors. Using translations between signatures, we can define morphisms between formulae, and therefore we can define interpretations between theories (in the standard sense). We deal with this issue in the next section.

Given a translation $\tau : L \to L'$ and given a L'-structure M, it is straightforward to define the restriction of $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{L}, \{\mathcal{P}^i \mid i \in I_0\} \rangle$ with respect to τ (or its reduct as it is called in model theory), as follows:

Definition 4.4. Given a translation $\tau : L \to L'$ and a L'-structure M we can define a L-structure $M|_\tau$ as follows:

- $\mathcal{W}|_\tau \stackrel{\text{def}}{=} \mathcal{W}$.
- $\mathcal{E}|_\tau \stackrel{\text{def}}{=} \mathcal{E} - \{e \mid e \in \mathcal{L}(\tau(\mathbf{U})) \cap \mathcal{L}'(\Delta'_0 - \tau(\Delta_0))\}$, where for any set of primitive actions S we define: $\mathcal{L}(S) \stackrel{\text{def}}{=} \bigcup\{\mathcal{L}(s) \mid s \in S\}$ and $\tau(S) = \bigcup\{\tau(s) \mid s \in S\}$.
- $\mathcal{L}|_\tau(a_i) \stackrel{\text{def}}{=} \{e \in \mathcal{L}(a) \mid e \in \mathcal{E}|_\tau\}$, for every $a_i \in \Delta'_0$.
- $\mathcal{L}|_\tau(p_i) \stackrel{\text{def}}{=} \mathcal{L}(\tau(p_i))$, for every $p_i \in \Phi'_0$.
- $\mathcal{R}|_\tau \stackrel{\text{def}}{=} \{w \xrightarrow{e} w' \in \mathcal{R} \mid e \in \mathcal{E}|_\tau\}$.
- $\mathcal{P}^i|_\tau(w, e) \Leftrightarrow P^{\tau(i)}(w, e)$.
- $w_0|_\tau \stackrel{\text{def}}{=} w_0$.

It is worth noting that the restriction of a standard structure of L' can be a non-standard structure of L. Note also that we take out of the model those events which belong to translated actions and actions outside of the translation (see item 2 of Definition 4.4), i.e., we only keep those events which are obtained by executing only actions of L or those which are obtained by executing actions outside of L. Some restrictions added below ensure that no important property of the original model is lost when we take its reduct.

Translations between languages and restrictions between models define a functor which is used in Institutions [29] to define logical systems. Note that a restriction of a given structure could be a structure which is not a locus, i.e., the obtained semantic entity violates the notion of locality as explained above. Furthermore, perhaps the reduct of a model loses some important properties. For this reason, we introduce the concept of τ-locus structures. We define some requirements on translations; given a translation $\tau : L \to L'$, consider the following set of formulae of the form:

- $\langle \tau(\gamma)\rangle \top \to \langle \tau(\gamma) \sqcap \overline{a_1} \sqcap \cdots \sqcap \overline{a_n}\rangle \top$, where γ is an atom of the boolean term algebra Δ_0/Φ_{BA}, and $a_1, \ldots, a_n \in \Delta_0' - \tau(\Delta_0)$.

These formulae say that the execution of the actions of L when translated to L' are not dependent on any action of L'; we can think of this as an independence requirement, i.e., the actions of L when translated to L' keep their independence. This is an important modularity notion. In practice, this can be ensured by implementing the two components (which these languages describe) in different processes. We denote this set of formulae by $\mathsf{ind}(\tau)$. Another requirement (which is related to independence) is that the new actions in Δ_0' (those which are not translations of any action in L) do not add new non-determinism to the translated actions. This fact can be expressed by the set of formulae of the following form:

- $\langle \tau(\gamma)\rangle \tau(\varphi) \to [\tau(\gamma)]\tau(\varphi)$, for every atom γ of the boolean algebra of terms obtained from L, and formula φ of L.

For a given translation $\tau : L \to L'$, we denote this set of formulae by $\mathsf{atom}(\tau)$, since they reflect the fact that the atomicity of the actions in L is preserved by translation.

Definition 4.5. Given a translation $\tau : L \to L'$ and a L'-structure M, we say that M is a τ-locus iff $M \models \mathsf{ind}(\tau)$, $M \models \mathsf{atom}(\tau)$ and $M|_\tau$ is a locus structure for L.

That is, a locus structure with respect to a translation τ is a structure which respects the locality and independence of L. Let us investigate some properties of τ-locus models. The first property says that in $M|_\tau$ we have all the paths that are needed.

Property 4.4. *Given a τ-locus model M, for every full path π' of M such that $\pi' \succeq \pi[0..i]$ (where $\pi[0..i]$ is a subpath in $M|_\tau$), there is full path $\pi'' \succeq \pi[0..i]$ in M such that π'' also is a full path of $M|_\tau$ and for any formula $\tau(\varphi)$: $\pi', i, M \models \tau(\varphi)$ iff $\pi'', i, M \models \tau(\varphi)$.*

The next result says that, in any τ-locus model, silent steps preserve properties:

Property 4.5. *If $\tau : L \to L'$, and M is a L' structure which is a τ-locus, then if $\pi, i, M|_\tau \models \varphi$, and $\pi(i)$ is non-local, then $\pi, i+1, M|_\tau \models \varphi$.*

Another important property is that an execution of a τ-locus model $M|_\tau$ cannot diverge by non-local actions when there exists the possibility of executing a local action:

Property 4.6. *If $\tau : L \to L'$, and M is a L' structure which is a τ-locus, then if for a path π of $M|_\tau$ we have $\pi_i \xrightarrow{e} \pi_{i+1}$ where e is local for $M|_\tau$, then there is no π' such that $\pi' \succeq \pi[0..i]$ and from position i all the events of π' are non-local for $M|_\tau$.*

Let us prove that local properties are preserved by τ-locus structures:

Theorem 4.5. *Let $\tau : L \to L'$ be a translation and M an L'-structure. If M is a τ-locus, then for any full path π of $M|_\tau$, $\pi, i, M \models \tau(\varphi)$ iff $\pi, i, M|_\tau \models \varphi$, for every formulae φ of L.*

We have obtained a semantical characterization of structures which respect the local behaviour of a language with respect to a given translation. Because we wish to use deductive systems to prove properties over a specification, it is important to obtain some axiomatic way of characterizing this class of structures. For a given translation $\tau : L \to L'$, consider the following (recursive) set of formulae:

$$\{\tau(\varphi) \to [\overline{\tau(\mathbf{U})}]\tau(\varphi) \mid \varphi \in \Phi'\}.$$

Roughly speaking, this set of axiom schemes says that if an action of an external component is executed, then the local state of the current module is preserved. We need other axioms to express the property that when we embed a module inside another part of the system, we want to ensure that the behaviour of the smaller module is preserved, in the following sense: we can introduce external events in some way in a given trace but we do not want that these external events add divergences that were not in the original trace. The following axiom does this: $\langle \tau(\mathbf{U})\rangle \top \to \mathsf{AFDone}(\tau(\mathbf{U}))$. This axiom expresses one of the conditions of local bisimulation, namely a trace cannot diverge by non-local events unless the component cannot execute any local action. It is worth noting that, if a local action is enabled in some state, then after executing a non-local action it will continue being enabled (as a consequence of the axiomatic schema described above), i.e., we require a fair scheduling of components, one which will not always neglect a component wishing to execute some of its actions.

Given a translation $\tau : L \to L'$, we denote this set of axioms, together with the axiomatic schema described above and the formulae $\mathsf{ind}(\tau)$ and $\mathsf{atom}(\tau)$, by $Loc(\tau)$. A nice property is that this set of formulae characterizes the L'-structures which are τ-loci.

Theorem 4.6. *Given a translation $\tau : L \to L'$, then a L'-structure M is a τ-locus iff $M \models Loc(\tau)$.*

We have presented an axiomatic system for an earlier version of this deontic logic in [7]. We need to add some axioms to that system to deal with the new operators introduced above. The axioms for the propositional part of the logic are:

1. The set of propositional tautologies.
2. A set of axioms for boolean algebras for action terms (a complete one), including standard axioms for equality.
3. The following set of axioms:

A1. $[\emptyset]\varphi$
A2. $\langle\alpha\rangle\varphi \wedge [\alpha]\psi \to \langle\alpha\rangle(\varphi \wedge \psi)$
A3. $[\alpha \sqcup \alpha']\varphi \leftrightarrow [\alpha]\varphi \wedge [\alpha']\varphi$
A4. $[\alpha]\varphi \to [\alpha \sqcap \alpha']\varphi$
A5. $\mathsf{P}^i(\emptyset)$, for every index i.
A6. $\mathsf{P}^i(\alpha \sqcup \beta) \leftrightarrow \mathsf{P}^i(\alpha) \wedge \mathsf{P}^i(\beta)$, for every index i.
A7. $\mathsf{P}^i(\alpha) \vee \mathsf{P}^i(\beta) \to \mathsf{P}^i(\alpha \sqcap \beta)$, for every index i.
A8. $\neg\mathsf{P}^i_{\mathsf{w}}(\emptyset)$, for every index i.
A9. $\mathsf{P}^i_{\mathsf{w}}(\alpha \sqcup \beta) \leftrightarrow \mathsf{P}^i_{\mathsf{w}}(\alpha) \vee \mathsf{P}^i_{\mathsf{w}}(\beta)$, for every index i.
A10. $\mathsf{P}^i_{\mathsf{w}}(\alpha \sqcap \beta) \to \mathsf{P}^i_{\mathsf{w}}(\alpha) \wedge \mathsf{P}^i_{\mathsf{w}}(\beta)$, for every index i.
A11. $\mathsf{P}^i(\alpha) \wedge \alpha \neq \emptyset \to \mathsf{P}^i_{\mathsf{w}}(\alpha)$, for every index i.
A12. $\mathsf{P}^i_{\mathsf{w}}(\gamma) \to \mathsf{P}^i(\gamma)$, where $[\gamma]$ is an atom in Δ_0/Φ_{BA} and for every index i.
A13. $\mathsf{O}^i(\alpha) \leftrightarrow \mathsf{P}^i(\alpha) \wedge \neg\mathsf{P}^i_{\mathsf{w}}(\overline{\alpha})$, for every index i.
A14. $[\alpha]\varphi \leftrightarrow \neg\langle\alpha\rangle\neg\varphi$
A15. $(a_1 \sqcup \cdots \sqcup a_n) =_{act} \mathbf{U}$
A16. $\langle\beta\rangle(\alpha =_{act} \alpha') \to \alpha =_{act} \alpha'$
A17. $\langle\gamma\rangle\varphi \to [\gamma]\varphi$, where $[\gamma]$ is an atom of Δ_0/Φ_{BA}
BA. $\varphi[\alpha] \wedge (\alpha =_{act} \alpha') \to \varphi[\alpha/\alpha']$
TempAx1. $\langle\mathbf{U}\rangle\top \to (\mathsf{AN}\varphi \leftrightarrow [\mathbf{U}]\varphi)$
TempAx2. $[\mathbf{U}]\bot \to (\mathsf{AN}\varphi \leftrightarrow \varphi)$
TempAx3. $\mathsf{AG}\varphi \leftrightarrow \neg\mathsf{E}(\top\ \mathcal{U}\ \neg\varphi)$
TempAx4. $\mathsf{E}(\varphi\ \mathcal{U}\ \psi) \leftrightarrow \psi \vee (\varphi \wedge \mathsf{ENE}(\varphi\ \mathcal{U}\ \psi))$
TempAx5. $\mathsf{A}(\varphi\ \mathcal{U}\ \psi) \leftrightarrow \psi \vee (\varphi \wedge \mathsf{ANA}(\varphi\ \mathcal{U}\ \psi))$
TempAx6. $[\bigsqcup_{a\in S} a \sqcap \alpha]\mathsf{Done}_S(\alpha)$
TempAx7. $[\bigsqcup_{a\in S} a \sqcap \overline{\alpha}]\neg\mathsf{Done}_S(\alpha)$
TempAx8. $\neg\mathsf{Done}_S(\emptyset)$
TempAx9. $\mathsf{B} \to \neg\mathsf{Done}_S(\alpha)$
TempAx10. $[\mathbf{U}]\neg\mathsf{B}$
TempAx11. $\mathsf{Done}_S(\alpha) \to [\overline{\bigsqcup_{a\in S} a}]\mathsf{Done}_S(\alpha)$
TempAx12. $\neg\mathsf{Done}_S(\alpha) \to [\overline{\bigsqcup_{a\in S} a}]\neg\mathsf{Done}_S(\alpha)$

4. the following deduction rules:
   - Standard rules for propositional logic.

TempRule1. if $\vdash \mathsf{B} \to \varphi$ and $\vdash \varphi \to [\mathbf{U}]\varphi$, then $\vdash \varphi$
TempRule2. if $\vdash \varphi$, then $\vdash \mathsf{AG}\varphi$
TempRule3. if $\vdash \varphi \to (\neg\psi \wedge \mathsf{EN}\varphi)\}$, then $\vdash \varphi \to \neg\mathsf{A}(\varphi'\ \mathcal{U}\ \psi)$
TempRule4. if $\vdash \varphi \to (\neg\psi \wedge \mathsf{AN}(\varphi \vee \neg\mathsf{E}(\varphi'\ \mathcal{U}\ \psi)))$, then $\vdash \varphi \to \neg\mathsf{E}(\vartheta\ \mathcal{U}\ \psi)$
TempRule5. if $\vdash \neg\mathsf{Done}(\mathbf{U}) \to \mathsf{AG}\varphi$, then $\vdash \varphi$

Axioms **A1**–**A17** are presented in [7] for the propositional version of this logic. Notice axiom **A13** which expresses the principle of replacement of equals for equals (the expression $\varphi[\alpha/\alpha']$ denotes the formula resulting from replacing α by α' in φ). Let us note axiom **A15**, this formula says that **U** is the non-deterministic choice between all the primitive actions (note that vocabularies contain a finite set of primitive actions). Axioms **TempAx1**–**TempAx2** relate the temporal and standard modalities, while axioms **TempAx4** and **TempAx5** are standard for CTL logics. Axioms **TempAx9** and **TempAx10** define the basic properties of the B predicate: they imply that no action was performed before, and after executing any action, B becomes false. Note that we also use B instead of $\neg\mathsf{Done}(\mathbf{U})$ in the induction rule. The rest of the axioms define the relativized Done() operator; note that in these axioms $\bigsqcup_{a\in S} a$ denotes the choice between all the actions in S. It is important to remark that in the case that $S = \Delta_0$ (i.e., when S is the set of all the primitive actions of the language), then the properties of $\mathsf{Done}_{\Delta_0}()$ are exactly those of the standard Done() operator as defined in [7]. We prove that this axiomatic system is sound with respect to locus structures.

Theorem 4.7. *The axiomatic system presented above is sound with respect to locus structures and the relation $\models$.*

We do not investigate the completeness of this system in this paper; let us just note that we have proved in [21] that this set of axioms defines a complete deduction system w.r.t. standard models without the relativized version of the Done() operator, and therefore it seems more or less straightforward to adapt that proof to the new semantics provided here.

In the following, by $\Gamma \vdash^L \varphi$ and $\vdash^L_S \varphi$ we denote two different situations. The first can be thought of as a "local" deduction relationship. This relationship holds when we have a proof, in the standard sense, of φ where some members of Γ may appear, but the only rule that we can apply to them is *modus ponens*. Alternatively, $\vdash^L_S \varphi$ says that, if we extend our axiomatic system with the formulae of S, then we can prove φ. An important difference is that the former notion of deduction preserves the deduction theorem. However, this theorem is not valid for the global version of deduction, an easy counterexample is: $\vdash_{S,\varphi} \mathsf{AG}\varphi$ (where S, φ is an abbreviation for $S \cup \{\varphi\}$). However, we can prove a variation of the deduction theorem:

Theorem 4.8. $\vdash^L_{S,\varphi} \psi$ *iff* $\vdash^L_S \mathsf{AG}\varphi \rightarrow \psi$.

Now we can prove that the deductive machinery obtained by adding the locality axioms preserves translations of properties. This fundamental property allows us to guarantee that components, when embedded in a wider system, conserve their properties.

Theorem 4.9. *Given a translation* $\tau : L \rightarrow L'$, *if* $\vdash^L \varphi$, *then* $\vdash^L_{Loc(\tau)} \tau(\varphi)$.

5. Defining components

A component is a piece of specification which is made up of a language and a set of axioms; these axioms describe the behaviour of the component and the extra assumptions about the component, e.g., independence and locality.

Definition 5.1. A component is a tuple $\langle L, A\rangle$ where: L is a language, as described in earlier sections, A is a set of axioms (the properties specified by the designers).

Given a component $C = \langle L, A\rangle$, we denote by $\vdash_C \varphi$ the assertion $\vdash^L_A \varphi$. A mapping between two components is basically an interpretation between the theory presentations that define them and defines a relationship of *being-part-of* as explained above.

Definition 5.2. A mapping $\tau : C \rightarrow C'$ between two components $C = \langle L, A\rangle$ and $C' = \langle L', A'\rangle$ is a translation $\tau : L \rightarrow L'$ such that: (i) $\vdash_{C'} \tau(\varphi)$, for every $\varphi \in A$. (ii) $\vdash_{C'} Loc(\tau)$.

It is worth noting that we require that the locality axioms must be theorems in the target component to ensure that the properties, including encapsulation and preservation of nondeterminism, of the smaller component are preserved. This is expressed by the following corollary.

Theorem 5.1. *If* $\tau : C \rightarrow C'$ *is a mapping between components* C *and* C', *then:* $\vdash_C \varphi \Rightarrow \vdash_{C'} \tau(\varphi)$.

Now that we have a notion of component, we need to have some way to put components together. We follow Goguen's ideas [11], where constructions coming from category theory are used to put together components of a specification. The same ideas are used in [9,13], where temporal theories are used for specifying pieces of concurrent programs, and translations between them are used for specifying the relationships between these components. The idea then is to define a category where the objects are components (specifications) and the arrows are translations between them; therefore, putting together components is achieved by using the construction of colimits. Of course, some prerequisites are required. Firstly, the category of components has to be finitely cocomplete and, secondly, the notion of deduction has to be preserved by translations (which is exactly what we proved above).

First, recall that the collection of all the languages and all the translations between them form the category **Sign**. Components and mappings between them also constitute a category.

Theorem 5.2. *The collection of all components and all the arrows between them form the category* ***Comp***.

The initial element of this category is the component with an empty language. Note that since the category of signatures is finitely cocomplete (its elements are just tuples of sets), the category of components is also finitely cocomplete; the forgetful functor from components to signatures reflects finite colimits (as shown for different logics in [29,30]).

Theorem 5.3. *The category* ***Comp*** *is finitely cocomplete.*

Putting together components is therefore achieved by taking the colimit of a given diagram of components; an important point here is that the colimit of a given diagram of specifications preserves the separation of deontic predicates. In the Section 8 we exhibit an example.

6. Reasoning about violations

In this section we introduce some formal machinery to capture some properties regarding violations. In particular, we are interested in studying those scenarios where the behaviour of one component may produce an error in other components.

Given a component C, we have a finite set $V = \{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$ of violation constants; these constants are intended to be used to identify the occurrence of faults. Each subset $S \subseteq V$ defines a possible set of violations that may happen

during a system execution, each of these sets can be defined by a predicate $*\mathtt{v}_1 \wedge \cdots \wedge *\mathtt{v}_n$, where $\mathtt{v}_1, \ldots, \mathtt{v}_n$ are the violation constants of the component being analysed and the $*$ denotes either a blank or $\neg$. These sets of violation predicates, together with the relationship $\subseteq$, form a lattice, note that during the execution of any component not all of these "states of violation" are reachable. Actually, the structure of the sets of violations that occur during the execution of a component, and the relationships between these sets, may not form a lattice; we introduce some formal machinery to reason about these structures. Furthermore, we study what happens with these structures when two or more components are put together.

For any predicate $V = *\mathtt{v}_1 \wedge \cdots \wedge *\mathtt{v}_n$ we define a set:

$$\mathcal{U} = \{\mathtt{v}_i \mid \mathtt{v}_i \text{ appears without negation in } V\}.$$

These sets induce an order $\leq_v$ over these kinds of predicates as follows:

$$V \leq_v V' \leftrightarrow \mathcal{U}(V) \supseteq \mathcal{U}(V').$$

Note that $\leq_v$ is contravariant to $\subseteq$; we can think of it as a relationship of improvement. For any component we want to establish which pairs $V \leq_v V'$ actually are possible for this component; in other words, we want to know from which error states we can recover or partially recover. To this end, we introduce the concept of *upgrading formula*.

Definition 6.1. Given a language L, the set of upgrading formulae for L is defined as follows:

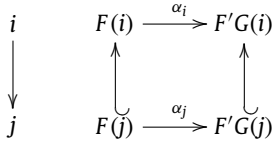
- If V and V' are violation predicates and $V <_v V'$, then,

  $$(V \rightarrow ([\alpha_1; \ldots; \alpha_n]V') \wedge (\langle\alpha_1; \ldots; \alpha_n\rangle\top)$$

  is an upgrading formula, where $\alpha_1, \ldots, \alpha_n$ are actions in L.

Here we define $[\alpha; \beta]\varphi \stackrel{\text{def}}{=} [\alpha][\beta]\varphi$. Note that an upgrading action expresses the idea that from a state with violations V we can upgrade to a state with violations V', where some violations or errors are no longer present. Note that the we require $V \rightarrow \langle\alpha_1; \ldots; \alpha_n\rangle\top$, this says that the upgrading action can be executed, otherwise any impossible action would be an upgrading action. Note that these formulae imply the temporal formula $V \rightarrow \mathsf{EF}V'$.

Let us consider a set $\mathcal{V} = \{\mathtt{v}_1, \mathtt{v}_2, \ldots\}$ containing all the possible violation predicates. We can form a category $\mathcal{C}(\mathcal{V})$ which has as objects the subsets of $\mathcal{V}$ and as arrows the functions between these sets. We want to use this category to capture in a categorical way the properties of upgrading actions. Consider the category **Pos** whose objects are partially ordered sets (which are categories) and whose morphisms are the functors between them (i.e., order preserving mappings). This category is complete and cocomplete [31]. We call a functor $F : I^{op} \rightarrow \mathcal{C}(\mathcal{V})$ (where I is a partially ordered set) an *upgrading diagram*. That is, for each object i of I, $F(i) \subseteq \mathcal{V}$, and for each arrow $i \rightarrow j$ in I, F maps it to an inclusion $F(j) \hookrightarrow F(i)$ in $\mathcal{C}(\mathcal{V})$. Now, a morphism between two upgrading diagrams $F : I^{op} \rightarrow \mathcal{C}(\mathcal{V})$ and $F' : J^{op} \rightarrow \mathcal{C}(\mathcal{V})$ is a functor (an order preserving mapping) $G : I^{op} \rightarrow J^{op}$ between I and J and a natural transformation $\alpha : F \stackrel{\cdot}{\rightarrow} F'G$. Naturality means that the following diagram commutes:



The category **Up** is the category whose objects are violation diagrams and whose arrows are pairs $\langle G : I \rightarrow J, \alpha : F \stackrel{\cdot}{\rightarrow} F'G\rangle : F \rightarrow F'$ as explained above. Since the category **Pos** and the category $\mathcal{C}(\mathcal{V})$ are finitely cocomplete, then for **Up** colimits can be calculated pointwise (see [32]); therefore **Up** is finitely cocomplete.

Theorem 6.1. *The category* **Up** *is cocomplete.*

Upgrading diagrams can be put together using colimits as is done with specifications. Given a component $C = \langle L, A, S\rangle$, we can define its upgrading diagram; it is a functor $U_C : I_C^{op} \rightarrow \mathcal{C}(\mathcal{V})$, where the elements of I_C are defined as follows. If V, V' are two violation states of C and $\vdash_C V \rightarrow ([\alpha_1; \ldots; \alpha_n]V') \wedge (\langle\alpha_1; \ldots \alpha_n\rangle\top)$ is a upgrading formula of C, then the pair $\langle V, V'\rangle$ is in I_C (and V, V' are elements of I_C). We also add the pairs $\langle V, V\rangle$ to satisfy reflexivity (and note that the defined relationship is transitive and antisymmetric). The functor $U_C : I_C^{op} \rightarrow \mathcal{C}(\mathcal{V})$ is defined as follows.

- For each violation state V which is an object of I_C, we have $U_I(V) = \mathcal{U}(V)$.
- For each arrow $V \rightarrow V'$ (pair) in I_C, it returns the inclusion $\mathcal{U}(V') \hookrightarrow \mathcal{U}(V)$ in $\mathcal{C}(V)$.

Note that we have equivalence classes of upgrading functions that represent the same transition between two violation states.

As remarked in Section 5, a system is made up of several components $C_1, \ldots, C_n$ and morphisms between them. They form a diagram (in the categorical sense) and therefore the final system is obtained by putting together the components using the colimit construction. Each of these components has a corresponding upgrading diagram, it seems natural to try

to build the upgrading diagram of the system from the upgrading diagrams of the components. However, when combining components it may be the case that the actions executed in one component introduce violations into the other components that interact with it. It is important to avoid scenarios where a recovery from an error in one component introduces a new error in other component, and therefore the system is always in a state of error. We study some conditions that allow us to establish an independence result to ensure that components do not introduce (via their behaviour) further violations in other components. To this end, we use the deontic predicates and their properties.

(Of course, the combination of the components may introduce new global faults that are not rooted in any fault resulting simply from the computation within a single component. But the characterization of such global faults and their amelioration are issues to be dealt with at the global level. At the moment, we are concerned with localizing the effect of local faults, providing guarantees for their insulation from the faults in other components.)

Consider the notions of permission and obligation in a deontic specification; we can expect that the execution of permitted actions does not introduce new violations into a state. This fact can be expressed by means of the following axiomatic schema:

$$\neg \mathsf{v}_\mathsf{i} \wedge \mathsf{P}^i(\alpha) \rightarrow [\alpha]\neg \mathsf{v}_\mathsf{i}.$$

This schema will allow us to reason about the interaction of components. A similar formula is proposed in [33] as an axiom of systems with deontic restrictions, the corresponding formula is called *GGG* since from a green state (a state without violations) and executing a green action (a permitted action) we can only reach green states. The property proposed above is stronger than Sergot's formula; because of this we use $SG(C)$, where C is a component, to refer to the formulae introduced above.

In the following, we investigate some scenarios where we can ensure some independence between the violations in the components. For the following theorems, we need to define formally what it means for two components to be coordinated via a variable or an action. Given a diagram $D : I \rightarrow \mathbf{Comp}$ with components $C_1, \ldots, C_n$, and colimit $\langle C, \tau_i : C_i \rightarrow C\rangle$, we say that two components C_i and C_j coordinate via an action c of C if we have an action c_i of C_i and an action c_j of C_j such that $\tau_i(c_i) = c = \tau_j(c_j)$, and we say that C_i and C_j coordinate via a variable p of C if there are variables p_i in C_i and p_j in C_j such that $\tau_i(p_i) = p = \tau(p_j)$.

Note that the $SG(C)$ predicate says that an execution of an allowed action cannot introduce a new violation into a state. Then, if we coordinate two components on actions which are always allowed (i.e., they are safe), then we can ensure that no violations are introduced when we execute a recovery (or a upgrading) action on one of the components. We need some extra notation to present these results. Given a language L, we say that $\mathbf{P}(\alpha)$ (α is in general allowed) iff $\mathsf{P}^1(\alpha) \wedge \cdots \wedge \mathsf{P}^n(\alpha)$ where $\{1, \ldots, n\}$ are the permission indexes of L, and we say that α is safe in a component C if $\vdash_C \mathbf{P}(\alpha)$.

Theorem 6.2. *Given a diagram $C_1 \leftarrow C \rightarrow C_2$, and the pushout of this diagram, denoted by $C_1 +_C C_2$, if (i) C_1 and C_2 do not coordinate via any violation, (ii) the actions in C when translated into actions of $C_1 +_C C_2$, say $c_1, \ldots, c_n$, are safe in $C_1 +_C C_2$ (i.e., $\vdash_{C_1+_CC_2} \mathbf{P}(c_i)$ for every i) and (iii) in the system axioms of C_1 (respectively, C_2) we have $SG(C_1)$ (respectively, $SG(C_2)$), then there is a morphism $\langle F, \alpha\rangle : U_{C_1} + U_{C_2} \rightarrow U_{C_1+_CC_2}$, such that all the components of α are iso and F is faithful.*

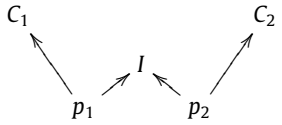
This theorem can be expressed by means of a slogan:

Coordination on safe actions is safe.

This property can be generalized to scenarios where we have a finite number of components and they only interact (or coordinate) by means of safe actions. Note that we require that components C_1 and C_2 do not coordinate via any violation. In the case that components coordinate via violations, the independence between the violation diagrams of each component are not respected any longer; it is possible that in this case the violation diagram of the system can be approximated using the colimits of the violation diagrams of the components. We do not investigate this in this paper. It is worth remarking that, in a concurrent setting, we want to keep the components as independent as possible, and coordination by means of violation constants may not be a good practice, to the extent that this is not strictly necessary.

It is important to analyse in detail what this theorem says. If we have two components and we put them together coordinating them via safe actions, then the violation state of one component does not affect the other component and vice versa. The isomorphism of the components of the natural transformation indicate that the number of violations is preserved in each violation state of the components, and the faithfulness of the functor indicates that the "shape" of the upgrading diagrams of each component is preserved by the upgrading diagram of the entire system.

However, the theorem described above can only be used in situations where components interact in a transparent way by identifying actions and shared variables. In some cases (see the example below) we will need a more complex interaction, in these cases, we need to use an interaction protocol; diagrammatically it has the form:



Here C_1 and C_2 are components, p_1 and p_2 are ports and I is the interaction protocol. Note that the part $p_1 \rightarrow I \leftarrow p_2$ of the diagram is a cospan; properties of cospans as interaction protocols are investigated in [34]. We can generalize Theorem 6.2 to those situations where interaction protocols are used.

Theorem 6.3. *Given a diagram $C_1 \leftarrow p_1 \rightarrow I \leftarrow p_2 \rightarrow C_2$, and the colimit of this diagram (whose tip is denoted by $C_1 +_I C_2$), if (i) I does not have violations. (ii) The actions I when translated to $C_1 +_I C_2$ are safe, then there is a morphism $\langle F, \alpha \rangle : U_{C_1} + U_{C_2} \rightarrow U_{C_1 +_I C_2}$, such that all the components of α are iso and F is faithful.*

Another scenario is when components interact via disjoint actions, i.e., this may happen when the interaction protocol just states some policy about the interaction but there is no hand-shaking communication. In these cases, we can use the following result:

Theorem 6.4. *Given a diagram $C_1 \leftarrow p_1 \rightarrow I \leftarrow p_2 \rightarrow C_2$, and the colimit of this diagram (whose tip is denoted by $C_1 +_I C_2$), if (i) for every action a in p_1 and action b in p_2 we have $\vdash_I [a \sqcap b]\top$, then there is a morphism $\langle F, \alpha \rangle : U_{C_1} + U_{C_2} \rightarrow U_{C_1 +_I C_2}$, such that all the components of α are iso and F is faithful.*

Notice that we can define *degrading* diagrams (i.e., diagrams that reflect the way in which violations are introduced during the execution of a component) in the same way that upgrading diagrams are defined, we only need to reverse the arrows in the definitions and theorems.

7. Related Work

Several frameworks have been proposed to modularize specifications; well-known examples are the B notation [35] or Object-Z [36]; however, these formalisms are designed to specify systems using pre and post-conditions, and therefore (as explained in [37]) they are not suitable to specify reactive or non-terminating systems. On the other hand, the modularization of temporal specifications has been investigated by Abadi and Lamport in their seminal paper [38]; in that work the authors use Temporal Logic of Actions (TLA) to specify systems, and a suitable logical machinery is introduced to enable compositional reasoning over specifications. Abadi and Lamport show that logical conjunction can be used to put together TLA components. In contrast to the formalism presented here, TLA is a linear temporal logic, and therefore some reasoning about branching time cannot be expressed in this logic; for example, the formula AGEFerror (*in every execution it is always possible to go into an error state*) is not expressible in linear temporal logics [39], and hence in TLA. Another difference between the logical framework presented above and TLA is the mechanism for structuring specifications: in our approach, components are expressed as theories, where logical morphisms between these theories are used to interconnect components; in this way the architectural design of the system is reflected in a categorical diagram; and therefore colimits can be used to obtain the final specification. As argued in [40], using morphisms to interconnect languages allows us not only to formally capture the notion of module, but also to capture the way in which these modules are structured. Note that morphisms allow us, for instance, to deduce the assumptions needed to ensure that an environment (i.e., the rest of the system) preserves the locality of components. Let us note that, when reasoning about open systems, we sometimes need to make assumptions about the environment. As noted by Abadi and Lamport [38,41], safety properties are easier to deal with than liveness properties; in particular, in the case of safety properties, composing proofs is a simple task. This is also true in our framework, notice that assumptions over the environment can be stated by formulae of the type $E \rightarrow P$, where E is a predicate over the input variables of the components (those modified by the environment). We do not distinguish between input and output variables in our logic, but it is straightforward to enrich our logic to include these distinctions. We remark that a better structuring of systems makes it possible to reason compositionally about their specifications; for instance; in the example shown below, the philosophers communicate with each other using a communication protocol (a fork); therefore, properties regarding philosophers can be proven in terms of their specification, and properties regarding the communication between philosophers can be proven using the specification of forks. Of course, sometimes when components are put together, we can obtain an inconsistent specification, for example, when the components require contradictory safety properties; an analysis of consistency is therefore needed when components are combined (for example, using the tableaux method described in [19]).

In this paper we have not distinguished between components and their interfaces; doing this would enable well-known structuring and abstraction techniques [42,43]. However, note that this distinction can be introduced in our framework: a component is made up of a language and a set of axioms; the former can be understood as the interface of the component; related notions such as ports, connectors, etc., can be defined using morphisms between languages, as is done in the example below. In this setting, interface specifications can be introduced by means of axioms; however, note that our logic is rather simple and additional operators or formalisms (e.g., interface automata [42]) may be needed in certain scenarios. The interested reader can consult [44] where these software engineering concepts are defined using categorical constructions, theories and morphisms.

Let us stress once again that we use a relationship of *being-part-of* between components to structure specifications; from the syntactical point of view, this relationship is captured by means of interpretations between logical theories [45]. From the semantical point of view, this relationship is formalized using the notions of *reduct* and *bisimulation*: roughly speaking, a relationship of *being-part-of* exists between a model of a component and a model of a system when the reduct of the latter is bisimilar to the former; intuitively, this says that the component is part of the system's behaviour. (We use the word "*system*" to refer to a wider module possibly made up of other components and an environment.) Bisimulation is a standard notion used for analysing the expressive power of modal/temporal logics and to relate models at different levels of abstraction; for example, in [46] the so-called Hennessy–Milner logic (HML) is introduced together with two suitable

notions of bisimilarity between models: strong and weak bisimulation (or observational equivalence); the former is used to relate labelled transition systems without silent actions, while the latter takes into account these kinds of actions. Bisimilar models are proved to be indistinguishable by HML formulae. However, as shown in [47] (where branching bisimulation is introduced), weak bisimulation does not preserve the branching structure of transition systems. Note that Hennessy–Milner logic considers a modality per action; in contrast to our logic, combinations of actions are not considered in HML and its semantics is given by labelled transition systems (where labels are restricted to transitions). Other variations of bisimulation have been used for analysing the expressive power of temporal logics; for example, [25] provides two versions of bisimulation: *divergence blind stuttering equivalence* ($\approx_{dbs}$) and *divergence sensitive stuttering equivalence* ($\approx_s$); the former is proven to be in agreement with the equivalence induced by CTL-X (CTL without the next operator) when arbitrary executions over Kripke systems are considered, and the latter is proven to be in agreement with CTL-X equivalences when only maximal paths are considered in the definition of $\models$. Also, in that paper, also the structures L^2TS are introduced, these structures have labels in both the states and the transitions (similar to the structures used to give the semantics of the logic presented above); using these constructions the authors connect Kripke structures with LTSs (labelled transition systems), and therefore they prove that the relation $\approx_s$ is the same as branching bisimulation, a relation introduced in [48] which ignores silent actions while preserving the branching structure of the system. As remarked in Section 4, our notion of bisimulation is similar to divergence sensitive stuttering equivalence (and hence to branching bisimulation); although our notion of bisimulation requires a stronger condition on models, local options must be preserved, this is so since we are interested in local reasoning, and from a component's point of view, the silent actions are those executed by other parts of the system. This is in contrast to HML (and similar approaches), where silent actions are usually used to abstract from internal actions.

On the other hand, we have included deontic operators in our logic; these operators allow us to express prescriptions and therefore to separate the good behaviours from the erroneous ones. As shown in Section 6, the notion of encapsulation can also be applied to deontic operators, enabling some compositional reasoning about fault-tolerance. Several frameworks and languages have been used to reason about fault-tolerant systems. Many of these approaches are designed for reasoning at a low level of specification (e.g., the implementation level); examples are: [49–52]. Some authors have extended process algebras to reason about fault-tolerant processes, for instance: [53–55]; however, in these works no extension of Hennessy–Milner logic with a corresponding deduction system is provided to prove properties about these languages. Specific examples of fault-tolerant systems were analysed using formal languages such as: TLA, the Alloy language and the Event-B formalism (e.g., [56–59]); however, in these works the difference between correct, ideal behaviour and incorrect or unexpected behaviour is just stated using *ad-hoc* mechanisms.

A more comprehensive framework for reasoning about fault-tolerance is introduced in [60], where the language of TLA is augmented with the notion of faulty action to enable fault-tolerance reasoning; so fault-tolerant systems are obtained using program transformations and then it is possible to prove that a given specification/program tolerates certain faults. Note that in our approach we do not have a set of faulty actions. Instead, we prescribe the correct behaviour of the system; including scenarios where a system action is used in an incorrect way causing an error. Furthermore, in [60] components are structured using logical conjunction, as usual in TLA; this is different from the approach taken here, where the main structuring mechanism is the notion of morphism. The diagram made up of components and morphisms between them gives us a picture of the system's architecture and the relationships between the languages participating in the system. These diagrams enable useful analyses over the system architecture, for example, by observing the shared parts between components, and then, as shown in Section 6, using the high-level language of category theory and the properties of deontic operators to prove properties about the composition of fault-tolerant components.

Finally, we may note that the semantical structures used in this paper are similar to others that can be found in the literature, for example, the L^2TS structures presented in [25]. Other related structures are Modal Transition Systems [61] (these structures have required and allowed transitions), which are used as a formalism to specify sets of implementations. Note that in our structures we have two kinds of transitions; however, they are used to give the semantics of deontic operators and not as a mechanism to specify implementations. Let us stress once again that our specifications are made up of logical theories, allowing us to achieve a high level of abstraction, labelled structures are used to give the semantics of theories, and therefore they enable analyses such as consistency checks, counterexample generation, etc.

8. An example

Now, we show an example to illustrate the application of these theorems in practice. We revisit the example that we presented in [7]. This example is a variation of Dijkstra's dining philosophers. We add the possibility that philosophers get sick and therefore they may have to go to the bathroom. The new scenario occurs when a philosopher takes some forks with him. (Obviously the worst scenario is when a philosopher takes with him two forks.) Here we follow the main ideas introduced in [13] to modularize the design of the standard version of Dijkstra's dining philosophers; note that no deontic operators are used in the referenced work. We introduce some notation to reduce the number of axioms in the specification shown below. The expression $\alpha \rightsquigarrow \varphi$ (where α is an action and φ a formula) denotes the following formula: $(\neg\varphi \rightarrow [\overline{\alpha}]\neg\varphi) \wedge ([\alpha]\varphi)$. Intuitively, this formula says that the action α is the only one which sets φ to true. We also introduce the notation $\alpha \uparrow \mathrm{v}$ which denotes the formula: $\mathsf{F}^i(\alpha) \rightarrow [\alpha]\mathrm{v}$, where α is an action, i is any index and v is

$\mathbf{f_1}.\mathsf{B} \to \neg \mathtt{l.up?} \wedge \neg \mathtt{r.up?}$
$\mathbf{f_2}.(\mathtt{l.up} \rightsquigarrow \mathtt{l.up?}) \wedge (\mathtt{l.down} \rightsquigarrow \neg \mathtt{l.up?})$
$\mathbf{f_3}.(\mathtt{r.up} \rightsquigarrow \mathtt{r.up?}) \wedge (\mathtt{r.down} \rightsquigarrow \neg \mathtt{r.up?})$
$\mathbf{f_4}.\neg(\mathtt{l.up?} \wedge \mathtt{r.up?})$
$\mathbf{f_5}.\mathtt{l.up?} \leftrightarrow \langle \mathtt{l.down} \rangle \top$
$\mathbf{f_6}.\mathtt{r.up?} \leftrightarrow \langle \mathtt{r.down} \rangle \top$

Fig. 5. XFork specification

$\mathbf{p_1} : \mathsf{B} \to \neg \mathtt{v}_1 \wedge \neg \mathtt{v}_2 \wedge \mathtt{thk}$
$\mathbf{p_2} : \mathtt{thk} \veebar \mathtt{hungry} \veebar \mathtt{eating} \veebar \mathtt{bad}$
$\mathbf{p_3} : \mathtt{eating} \leftrightarrow \mathtt{has}_{\mathtt{L}} \wedge \mathtt{has}_{\mathtt{R}} \wedge \neg \mathtt{bad}$
$\mathbf{p_4} : \neg \mathtt{hungry} \to \mathsf{AF}\mathtt{hungry}$
$\mathbf{p_5} : \neg \mathtt{eating} \to \mathsf{P}^1(\mathbf{U})$
$\mathbf{p_6} : \mathtt{eating} \to \mathsf{O}^1(\mathtt{down}_{\mathtt{L}} \sqcap \mathtt{down}_{\mathtt{R}})$
$\mathbf{p_7} : \mathtt{down}_{\mathtt{L}} \uparrow \mathtt{v}_1 \wedge \mathtt{down}_{\mathtt{R}} \uparrow \mathtt{v}_2$
$\mathbf{p_8} : (\mathtt{down}_{\mathtt{L}} \rightsquigarrow \neg \mathtt{v}_1) \wedge (\mathtt{down}_{\mathtt{R}} \rightsquigarrow \neg \mathtt{v}_2)$
$\mathbf{p_9} : (\mathtt{getthk} \rightsquigarrow \mathtt{thk}) \wedge (\mathtt{getbad} \rightsquigarrow \mathtt{bad}) \wedge (\mathtt{gethungry} \rightsquigarrow \mathtt{hungry})$
$\mathbf{p_{10}} : \mathtt{thk} \to \mathtt{down}_{\mathtt{L}} \wedge \mathtt{down}_{\mathtt{R}}$
$\mathbf{p_{11}} : \mathtt{hungry} \to \mathtt{down}_{\mathtt{L}} \wedge \mathtt{down}_{\mathtt{R}}$
$\mathbf{p_{12}} : \mathtt{hungry} \wedge \langle \mathtt{up}_{\mathtt{L}} \sqcup \mathtt{up}_{\mathtt{R}} \rangle \top \to \mathsf{AN}\mathtt{eating} \vee \mathsf{AN}\mathtt{hungry}$
$\mathbf{p_{13}} : \mathtt{bad} \to [\overline{\mathtt{getthk}}]\mathtt{bad}$
$\mathbf{p_{14}} : \mathtt{eating} \to \mathsf{AN}(\mathtt{thk} \vee \mathtt{bad})$

Fig. 6. Phil specification

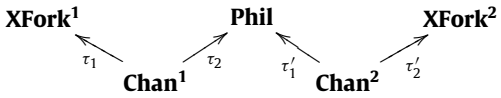


Fig. 7. Putting together forks with philosophers

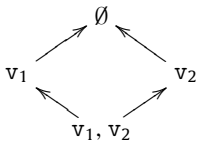
a violation predicate. This formula can be read as saying that some executions of α cause a violation of type v. Further notation can be introduced to obtain a higher level specification language, we leave this for further work.

First, let us consider the specification of a fork. The language of a fork has the following actions: $\Delta_0 = \{\mathtt{l.up}, \mathtt{l.down}, \mathtt{r.up}, \mathtt{r.down}\}$, the following predicates: $\Phi_0 = \{\mathtt{l.up?}, \mathtt{r.up?}\}$ and no violations. Intuitively, we have two ports by means of which we can use the forks; one is for the left philosopher and the other one is for the right philosopher. Note that this implies that the philosophers do not coordinate directly via any action (also note that these actions are mutually disjoint). The axioms of the fork are shown in Fig. 5. As explained above, a fork can be held onto by the philosopher on the left or by the philosopher on the right. Therefore, we have two actions that reflect this action: l.up and r.up. Obviously they are disjoint (as stated by axiom $\mathbf{f_4}$), meaning that only one of the philosophers can be holding onto the fork.

The specification for a philosopher is shown in Fig. 6. The actions of the specification are the following: {getthk, getbad, gethungry, $\mathtt{up}_{\mathtt{L}}$, $\mathtt{up}_{\mathtt{R}}$, $\mathtt{down}_{\mathtt{L}}$, $\mathtt{down}_{\mathtt{R}}$}. The action getthk indicates when the philosopher goes to the *thinking* state, getbad takes the philosopher to the *sick* state. The action gethungry takes a philosopher from the *thinking* state to the *hungry* state. The actions $\mathtt{up}_{\mathtt{L}}$ and $\mathtt{up}_{\mathtt{R}}$ are used for the philosopher to take the left or right fork, respectively. The predicates of the component are the following: {$\mathtt{has}_{\mathtt{L}}$, $\mathtt{has}_{\mathtt{R}}$, thk, eating, hungry, bad}. In addition, we have two violations $\{\mathtt{v}_1, \mathtt{v}_2\}$. We also consider the predicates: $\neg \mathtt{v}_i \wedge \mathsf{P}^1(\alpha) \to [\alpha] \neg \mathtt{v}_i$ as belonging to the axioms, for every i. Most of the axioms a self-explanatory, we discuss the remaining axioms. Axiom $\mathbf{P_4}$ says that a philosopher who is thinking will become hungry in the future; axiom $\mathbf{P_5}$ states that, when the philosopher is not eating, then everything is allowed. Axioms $\mathbf{p_7}$-$\mathbf{p_8}$ specify how the violations occur in a given execution of this specification and which are the recovery actions. Note that, in axiom $\mathbf{P_6}$, we say that, if a philosopher is eating, then it will be obliged to return both forks. We simplify the problem by requiring that philosophers can only eat for a unit of time (axiom $\mathbf{p_{14}}$).

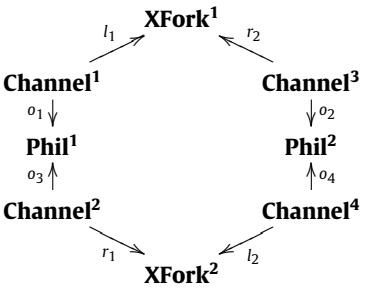
Suppose that we want to obtain the specification of a unique philosopher with two forks. We need to define some way of connecting the different components. With this goal in mind, we define a component **Chan** which only has actions $\{\mathtt{port}_1, \mathtt{port}_2\}$ with no predicates and no violations. Using channels, we can connect the forks with the philosopher taking the colimit of the diagram shown in Fig. 7. The components $\mathbf{XFork^1}$ and $\mathbf{XFork^2}$ are "instances" of the specification **XFork** (i.e., they obtained from **XFork** by renaming the symbols using the subindex i), and $\mathbf{Chan^1}$ and $\mathbf{Chan^2}$ are "instances" of **Chan**. Here τ_1 : **Chan** $\to$ **XFork** maps $\mathtt{port}_1^1 \mapsto$ lup and $\mathtt{port}_2^1 \mapsto$ ldown, whereas τ_2 : **Chan** $\to$ **Phil** maps $\mathtt{port}_1^1 \mapsto \mathtt{up}_{\mathtt{L}}$ and $\mathtt{port}_2^1 \mapsto \mathtt{down}_{\mathtt{L}}$ and similarly for τ_1' and τ_2'. In other words, these morphisms connect the right and the left fork with the philosopher. Notice that in $\mathbf{XFork^1}$, $\mathbf{XFork^2}$ and **Phil** we consider the necessary axioms of locality required by the corresponding morphisms.

Let us call the colimit object of this specification **FPhil**, where the morphism f_1 : **XFork** $\to$ **FPhil**, f_2 : **XFork** $\to$ **FPhil**, p_1 : **Phil** $\to$ **FPhil**, c_1 : **Chan** $\to$ **FPhil** and c_2 : **Chan** $\to$ **FPhil**, are the required morphisms from the base of the cocone to the colimit object. Now, the upgrading diagram of **FPhil** is as shown in Fig. 8. The formulae at the right in this figure indicate the properties that we need to prove to show that this diagram is correct. Intuitively, the worst state is when a philosopher is in the bathroom with both forks. He can recover from this scenario by putting one of the forks down, and then he can go into a normal state by putting the other fork down. To prove the arrow from $\mathtt{v}_1$ to $\emptyset$, we proceed as follows.



- $\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \wedge \mathtt{v}_2 \rightarrow [\mathtt{down}_\mathtt{L}]\neg\mathtt{v}_1 \wedge \mathtt{v}_2$
- $\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \wedge \mathtt{v}_2 \rightarrow [\mathtt{down}_\mathtt{R}]\mathtt{v}_1 \wedge \neg\mathtt{v}_2$
- $\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \wedge \neg\mathtt{v}_2 \rightarrow [\mathtt{down}_\mathtt{L}]\neg\mathtt{v}_1 \wedge \neg\mathtt{v}_2$
- $\vdash_{\mathbf{FPhil}} \neg\mathtt{v}_1 \wedge \mathtt{v}_2 \rightarrow [\mathtt{down}_\mathtt{R}]\neg\mathtt{v}_1 \wedge \neg\mathtt{v}_2$

Fig. 8. Upgrading diagram of **FPhil**



$$
\begin{aligned}
l_1 &= \{\mathtt{port}_1^1 \mapsto \mathtt{l.up}, \mathtt{port}_2^1 \mapsto \mathtt{l.down}\}\\
o_1 &= \{\mathtt{port}_1^1 \mapsto \mathtt{up}_\mathtt{L}, \mathtt{port}_2^1 \mapsto \mathtt{down}_\mathtt{L}\}\\
l_2 &= \{\mathtt{port}_1^4 \mapsto \mathtt{l.up}, \mathtt{port}_2^4 \mapsto \mathtt{l.down}\}\\
o_4 &= \{\mathtt{port}_1^4 \mapsto \mathtt{up}_\mathtt{L}, \mathtt{port}_2^4 \mapsto \mathtt{down}_\mathtt{L}\}\\
r_1 &= \{\mathtt{port}_1^2 \mapsto \mathtt{l.up}, \mathtt{port}_2^1 \mapsto \mathtt{l.down}\}\\
o_3 &= \{\mathtt{port}_1^2 \mapsto \mathtt{up}_\mathtt{R}, \mathtt{port}_2^2 \mapsto \mathtt{down}_\mathtt{R}\}\\
r_2 &= \{\mathtt{port}_1^3 \mapsto \mathtt{r.up}, \mathtt{port}_2^4 \mapsto \mathtt{r.down}\}\\
o_4 &= \{\mathtt{port}_1^3 \mapsto \mathtt{up}_\mathtt{R}, \mathtt{port}_2^4 \mapsto \mathtt{down}_\mathtt{R}\}
\end{aligned}
$$

Fig. 9. Two philosophers eating

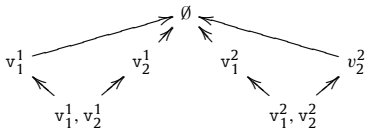


Fig. 10. Coproduct of upgrading diagrams.

1.	$\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \rightarrow \neg\mathtt{eating}$	Property of **FPhil**
2.	$\vdash_{\mathbf{FPhil}} \neg\mathtt{eating} \rightarrow \mathsf{P}^1(\mathbf{U})$	$\mathbf{p_5}$
3.	$\vdash_{\mathbf{FPhil}} \mathsf{P}^1(\mathbf{U}) \rightarrow \mathsf{P}^1(\mathtt{down}_\mathtt{L})$	DPL
4.	$\vdash_{\mathbf{FPhil}} \neg\mathtt{v}_2 \wedge \mathsf{P}^1(\mathtt{down}_\mathtt{L}) \rightarrow [\mathtt{down}_\mathtt{L}]\neg\mathtt{v}_2$	*SG*(**Phil**)
5.	$\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \wedge \neg\mathtt{v}_2 \rightarrow [\mathtt{down}_\mathtt{L}]\neg\mathtt{v}_2$	ML, 1, 2, 4
6.	$\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \wedge \neg\mathtt{v}_2 \rightarrow [\mathtt{down}_\mathtt{L}]\neg\mathtt{v}_1 \wedge \neg\mathtt{v}_2$	PL, $\mathbf{p_9}$, 5
7.	$\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \rightarrow \mathtt{has}_L$	Property of **Phil**
8.	$\vdash_{\mathbf{XFork}} \mathtt{lup?} \rightarrow \langle\mathtt{ldown}\rangle\top$	$\mathbf{f_5}$
9.	$\vdash_{\mathbf{FPhil}} \mathtt{has}_\mathtt{L} \rightarrow \langle\mathtt{down}_\mathtt{L}\rangle\top$	Def.f_1 & Theorem 5.1
10.	$\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \rightarrow \langle\mathtt{down}_\mathtt{L}\rangle\top$	PL, 7, 9
11.	$\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \wedge \neg\mathtt{v}_2 \rightarrow \langle\mathtt{down}_\mathtt{L}\rangle\top$	PL, 10
12.	$\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \wedge \neg\mathtt{v}_2 \rightarrow [\mathtt{down}_\mathtt{L}](\neg\mathtt{v}_1 \wedge \neg\mathtt{v}_2) \wedge \langle\mathtt{down}_\mathtt{L}\rangle\top$	PL,10, 6

In this proof, the acronym DPL means that we can obtain the corresponding line using basic properties of the logic, similarly for PL (propositional logic) and ML (modal logic). Note that, in line 4, we use the *SG*(**Phil**) property. In lines 1 and 7 we have used some basic properties of the specification that can be proven straightforwardly. The other transitions between violation states can be proven in a similar way.

We can build a complete specification with forks and philosophers interacting. Let us keep this simple and consider only two philosophers. We can use the channels to coordinate the two philosophers. Consider the diagram shown in Fig. 9. The colimit of this diagram gives us the final design (say **TPhils**), and note that the colimit produces the corresponding specification with all the needed renaming of clashing symbols. Note that at the right of this figure the different mappings appearing in the diagram are defined. These mappings define how the different parts of the design interconnect (as explained in [13]). The interesting point here is to analyse what happens with the upgrading diagram in this system, when we add an extra philosopher. Note that the two instances of **Phil** do not coordinate via any action (both coordinate with **XFork**, but using different channels), and therefore Theorem 6.4 can be applied here, obtaining that this specification preserves the coproduct of the upgrading diagrams of each philosopher. Note that the coproduct of the upgrading diagrams of each philosopher with forks is the one illustrated in Fig. 10. This means that each of the (formulae which act as witnesses of a) transition of this diagram can be proven from the specification **TPhils**. It is worth investigating if there are other transitions

(since the theorem above says that we have a faithful (injective) functor, however we cannot ensure that this functor is full (surjective)). Note that when we have two philosophers, if one of them has a fork, then the other cannot start eating, and therefore there is no way to reach a state violation of the type $\mathrm{v}_1^1 \wedge \mathrm{v}_2^2$. This kind of extra-transition depends on how many philosophers we have in this specification; for example, if we have three philosophers, we can obtain further violation states.

Summarizing, Theorems 6.2 and 6.4 allow us to deduce some basic transitions between violation states. However, some other transitions could be dependent on the specification being developed and have to be investigated by the designer (although it is worth noting that these theorems give us a good starting point to analyse the violation structure of a specification built from several components).

9. Further remarks

In this paper we have introduced a basic framework to modularize temporal and deontic specifications. The main idea is to use a notion of bisimulation to capture the concept of encapsulation or locality, which is, obviously, related to the concept of module or component. In contrast to the work of Fiadeiro and Maibaum [13], where a linear time logic is used, the main formalism used in this paper is a branching time logic, in which non-determinism is naturally reflected. The notion of encapsulation is important when reasoning about components; it allows us to use important deduction rules, such as the induction rule introduced in Section 3, which facilitates the compositional reasoning about specifications.

In addition, we have provided deontic predicates which can be used to specify ideal behaviour of systems, and therefore to model some concepts related to fault-tolerance; some examples and motivations are described in [21]. The novel part of the deontic logic presented here is the stratified levels of permissions, prohibitions and obligations, which enables us to avoid having global normative constraints (i.e., the normative restrictions imposed in a component do not affect the other components in the system). These stratified levels of permission can also be used to express different levels of ideal behaviours. Violation predicates allow us to capture the scenarios where deontic predicates are not fulfilled. We have proposed a basic formalism to reason about these violations; this framework allows us to state some properties about the composition of components and the preservation of recovery actions, modulo some deontic properties. We believe that these kinds of properties will make easier the analyses of specifications of fault-tolerant systems.

We presented a simple example to illustrate the use of these ideas in practice. In this example, a variation of Dijkstra's philosophers, we use deontic predicates to state what the ideal behaviour of philosophers are, the specification is built from in several components and they are then used (together with morphisms) to obtain the final design. Finally, it is worth mentioning that the logic is decidable and we have proposed a tableaux system for this logic in [19]; this will enable us to perform automatic analysis of specifications. We leave this as further work.

Appendix. Proofs

Proof of Theorem 4.1. If from position i in π we have an infinite sequence of non-local events, then $\pi_i \stackrel{\infty}{\Rightarrow}$ and therefore, by definition of local bisimulation, $\pi'_j \stackrel{\infty}{\Rightarrow}$. Thus we have some full path π_2 such that $\pi Z \pi_2$. Otherwise, we have some e and k such that $\pi_i \stackrel{e}{\Rightarrow} \pi_k$ in π; but, since $\pi_i Z \pi'_j$ we can find a state $v_{k'}$ in M' such that $\pi'_j \stackrel{e}{\Rightarrow} v_{k'}$. We denote by $\pi''[0..k]$ the extension of $\pi'[0..j]$ obtained by adding the path above. Then, we have $\pi[0..k] Z \pi''[0..k']$. Thus, for any extension of $\pi[0..i]$, we can find a corresponding extension of π', and therefore take π_2 to be the maximal such extension and we have $\pi Z \pi_2$. □

Proof of Property 4.1. The proof is by induction; the basis is straightforward. For the inductive case, suppose that $\pi_i Z \pi'_{f_\pi(i)}$; if $\pi_i \stackrel{e_i}{\rightarrow} \pi_{i+1}$ and e_i is non-local, then $f_\pi(i+1) = i$ and $\pi_{i+1} Z \pi'_{f_\pi(i)}$ by definition of local bisimulation. Otherwise, $f_\pi(i+1) = \min_{>f_\pi(i)}(\mathrm{Loc}(\pi'))$, and by definition of bisimulation between paths we get $\pi_{i+1} Z \pi_{f_\pi(i+1)}$. □

Proof of Property 4.2. The proof is by induction on i; the basis is straightforward: $\#Loc(\pi[0..0]) = 0 = \#Loc(\pi'[0..f_\pi(i)])$. For the inductive case: suppose that: $\#Loc(\pi[0..i]) = \#Loc(\pi[0..f_\pi(i)])$. Then, if $\pi_i \stackrel{e_i}{\rightarrow} \pi_{i+1}$ in π and e_i is non-local, then $\#Loc(\pi[0..i+1]) = \#Loc(\pi[0..i])$, and then $f_\pi(i+1) = f_\pi(i)$ and $\#Loc(\pi[0..i+1]) = \#Loc(\pi[0..f_\pi(i+1)])$. If e_i is local, then $\#Loc(\pi[0..i+1]) = \#(\mathrm{Loc}(\pi[0..i]) \cup \{e_i\})$ and then we have $\pi'_{f_\pi(i)} \stackrel{e_i}{\Rightarrow} \pi'_{f_\pi(i+1)}$, and then $\#Loc(\pi'[0..f_\pi(i+1)]) = \#(\mathrm{Loc}(\pi'[0..f_\pi(i)]) \cup \{e_i\}) = \#Loc(\pi[0..i+1])$. □

Proof of Theorem 4.2. The proof is by induction on φ.
Base Case. We know $L(\pi_i) = L(\pi'_{f_\pi})$, which implies that $\pi, i, M \vDash p_j$ iff $\pi', f_\pi(i), M' \vDash p_j$. The proof is similar for equations and deontic predicates. For the $\mathsf{Done}_S()$ operator, suppose that $\pi, i, M \vDash \mathsf{Done}_S(\alpha)$, then, for $k = \max_{<i}(\mathrm{Loc}_S(\pi))$, we have that $e_k \in \mathcal{A}(\alpha)$, and $\pi_{k-1} Z \pi'_{f_\pi(k-1)}$. But then we have $\pi'_{f_\pi(k-1)} \stackrel{e}{\Rightarrow} \pi'_{f_\pi(k)}$, and $\pi'_{f_\pi(k)} \stackrel{\epsilon}{\Rightarrow} \pi'_{f_\pi(i)}$, thus $\pi', f_\pi(i), M' \vDash \mathsf{Done}_S(\alpha)$.
Ind. Case. If $\pi, i, M \vDash [\alpha]\varphi$, then suppose $\pi', f_\pi(i), M' \nvDash [\alpha]\varphi$. Then, for some $\pi_2 \succeq \pi'[0..f_\pi(i)]$, we have a $k = \min_{>i}(\mathrm{Loc}(\pi_2))$ such that $(\pi_2)_k \in \mathcal{A}(\alpha)$ and $\pi_2, k, M' \nvDash \varphi$. By Theorem 4.1, we know that we have a full path $\pi_1 \succeq \pi[0..i]$ such that $\pi_1 Z \pi_2$. By the definition of bisimulation between paths we know that if $(\pi_2)_{f_{\pi_1}(i)} \stackrel{e}{\Rightarrow} (\pi_2)_k$ in π_2, then

$(\pi_1)_i \stackrel{e}{\Rightarrow} (\pi_1)_{f_{\pi_2}(k)}$ in π_1. Applying induction on the symmetric statement of the theorem, we get $\pi_2, f_{\pi_2}(k), M \not\models \varphi$ which is a contradiction. The other direction is similar.

If $\pi, i, M \models \mathsf{AN}\varphi$ the argument is as above.

If $\pi, i, M \models \mathsf{A}(\varphi\ \mathcal{U}\ \psi)$, suppose $\pi', f_\pi(i), M' \not\models \mathsf{A}(\varphi\ \mathcal{U}\ \psi)$. Then, if $\pi', f_\pi(i), M' \not\models \varphi$, we get a contradiction. Otherwise, we must have a full path $\pi_2 \succeq \pi'[0..f_\pi(i)]$, such that for every $j \in Loc(\pi_2)$ we have $\pi_2, j, M' \not\models \psi$. Now, as explained above, we have a $\pi_1 Z \pi_2$ and for this π_1 we have a $k \in Loc(\pi_1)$ such that $\pi_1, k, M \models \psi$, for this k we have that $\pi_2, f_\pi(k), M' \models \psi$, by induction. But note that $\pi_2(f_\pi(k)) \in Loc(\pi_2)$ which gives us a contradiction, and therefore $\pi', f_\pi(i), M' \models \mathsf{A}(\varphi\ \mathcal{U}\ \psi)$. The other direction is similar.

If $\pi, i, M \models \mathsf{E}(\varphi\ \mathcal{U}\ \psi)$, and suppose $\pi', f_\pi(i), M' \not\models \mathsf{E}(\varphi\ \mathcal{U}\ \psi)$, then if $\pi', f_\pi(i), M' \not\models \varphi$ and $\pi', f_\pi(i), M', \not\models \psi$, we get a contradiction. Otherwise, we have that for every path $\pi'' \succeq \pi'[0..f_\pi(i)]$ and for every $k \in Loc(\pi_2^{f_\pi(i)})$, $\pi'', k, M' \not\models \psi$ holds . Note that we have a π_2 in M' such that $\pi_1 Z \pi_2$ (where π_1 is that full path mentioned above), and then by induction $\pi_2, f_\pi(i), M' \models \psi$. Furthermore, note that $\pi_2(f_\pi(i))$ is a local event, which contradicts the assumption above, and therefore $\pi', f_\pi(i), M' \models \mathsf{E}(\varphi\ \mathcal{U}\ \psi)$. The other direction is similar. □

Proof of Property 4.4. The proof is direct using the properties of independence and atomicity. □

Proof of Property 4.5. The proof is by induction on φ; the cases are straightforward using the properties of local bisimulation, and the fact that a τ-locus model is bisimilar to a standard model. □

Proof of Property 4.6. Suppose that we have such a path; then, since $M|_\tau$ is bisimilar to a standard model, we can bisimulate the path π' until i. Thus, we have some state v in the standard model such that $\pi_i Z v$, but from there π' diverges with non-local events, and therefore there is no way to bisimulate it. In addition, v has a successor since π_i has a successor reachable by local events. From here we obtain that M is not a τ-locus model, which is a contradiction. □

Proof of Theorem 4.5. The proof is by induction on φ.

Base Case. It is straightforward using the definition of $M|_\tau$.

Ind. Case. If $\pi, i, M \models \tau([\alpha]\varphi)$ which is equivalent to $\pi, i, M \models [\tau(\alpha)]\tau(\varphi)$, and now suppose that $\pi, i, M|_\tau \not\models [\alpha]\varphi$. From here we have that there exists a path $\pi' \succeq \pi[0..i]$ such that $\pi, i+1, M|_\tau \not\models \varphi$, where $\pi'(i+1) \in \mathcal{U}|_\tau(\alpha)$. Now we have the same trace in M, which gives us a contradiction by induction. If $\pi, i, M|_\tau \models [\alpha]\varphi$, suppose $\pi, i, M \not\models [\tau(\alpha)]\tau(\varphi)$, and then we have a $\pi' \succeq \pi[0..i]$ (noting that, if π' is not a full path of $M|_\tau$ then, applying Property 4.4, we can find an equivalent path which belongs to this model) such that $\pi, i, M \not\models \tau(\varphi)$ and $\pi'(i) \in \mathcal{U}(\tau(\alpha))$. By definition of reduction, we have that $\pi'(i) \in \mathcal{U}|_\tau(\alpha)$, which applying induction, gives us a contradiction, and therefore $\pi, i, M \models [\tau(\alpha)]\tau(\varphi)$.

Suppose $\pi, i, M \models \tau(\mathsf{AN}(\varphi))$ and $\pi, i, M|_\tau \not\models \mathsf{AN}\varphi$. Then, if i is the last position of π, then we have $\pi, i, M|_\tau \not\models \varphi$, which gives us a contradiction, since by induction this implies $\pi, i, M \not\models \varphi$. If i is not the last position of π, then, for $k = \min_{>i}(\mathrm{Loc}_L(\pi))$, we have $\pi, k, M|_\tau \not\models \varphi$, note that $\pi_i \stackrel{e}{\Rightarrow} \pi_k$ where e is local for L, and then in M we have that it is the next position where an event of $a_1 \sqcup \cdots \sqcup a_n$ is executed, and therefore $\pi, k, M \not\models \mathsf{Done}(a_1 \sqcup \cdots \sqcup a_n) \rightarrow \varphi$, which contradicts what we said above, and therefore $\pi, i, M|_\tau \models \mathsf{AN}\varphi$. The other direction is similar.

Suppose that $\pi, i, M \models \tau(\mathsf{A}(\varphi\ \mathcal{U}\ \psi))$ and $\pi, i, M|_\tau \not\models \mathsf{A}(\varphi\ \mathcal{U}\ \psi)$. If $\pi, i, M|_\tau \not\models \varphi$ and $\pi, i, M|_\tau \not\models \psi$ (and the same reasoning is applied when φ and ψ are not true at some moment before ψ comes true), then by induction we obtain a contradiction. If, for some $\pi' \succeq \pi[0..i]$, we have that $\pi', k, M|_\tau \not\models \psi$, for every $k \in \mathrm{Loc}_L(\pi^i)$, then note that for this π' in M we have $\pi, i, M \not\models \tau(\psi)$ (by induction) and from here if a position $j \leq i$ is reached by a non-local event for L we have, by Property 4.5 and induction, that $\pi', j, M \models \tau(\psi)$, and if it is local, then we have by the supposition above that $\pi, j, M \models \tau(\psi)$, i.e., for every $j \geq i$ $\pi, j, M \not\models \tau(\psi)$, which contradicts our initial assumption, and therefore $\pi, i, M|_\tau \models \mathsf{A}(\varphi\ \mathcal{U}\ \psi)$. The other direction is similar.

If $\pi, i, M \models \tau(\mathsf{E}(\varphi\ \mathcal{U}\ \psi))$, then we have some $\pi' \succeq \pi[0..i]$ such that there is a k such that $\pi, k, M \models \tau(\psi)$ where $k \geq i$, and for every $j \in \mathrm{Loc}_{L'}(\pi')$ such that $i \leq j \leq k$, we have $\pi', j, M \models \tau(\varphi)$; then, since $\mathrm{Loc}_L(\pi') \subseteq \mathrm{Loc}_{L'}(\pi')$ and using induction, we have that for every $j \leq k$ such that $j \in \mathrm{Loc}_L(\pi')$, $\pi, j, M|_\tau \models \varphi$, and $\pi, k, M|_\tau \models \psi$. Now if $k \notin \mathrm{Loc}_L(\pi')$, then, by Property 4.4, it must be a $k' \in Loc(\pi')$ such that $k' \leq k$ and $\pi', k', M|_\tau \models \psi$. On the other hand, if $\pi, i, M|_\tau \models \mathsf{E}(\varphi \in \psi)$, then we have some $\pi' \succeq \pi[0..i]$ such that $\pi', k, M|_\tau \models \psi$, where $k \in \mathrm{Loc}_L(\pi')$, and for every $i \leq j \leq k$ with $j \in \mathrm{Loc}_L(\pi')$ we have $\pi', j, M|_\tau \models \varphi$. Note that, using Property 4.4, we have that for every position $j \in \mathrm{Loc}_{L'}(\pi')$ such that $i \leq j \leq k$, we have that $\pi', j, M \models \tau(\varphi)$ (since, if it is a local event for L, we show above that it satisfies φ, otherwise it preserves the property), and $k \in \mathrm{Loc}_{L'}(\pi')$ and then $\pi, k, M \models \psi$ by induction. □

Proof of Theorem 4.6. First, let us prove that, if M is a τ-locus, then it satisfies $Loc(\tau)$. By definition it satisfies $\mathsf{ind}(\tau)$ and $\mathsf{atom}(\tau)$), and by Property 4.4 and Theorem 4.5 we have that the model satisfies the axiomatic schema. On the other hand, note that the other axiom is satisfied since we require that M is local bisimilar to a standard model, and therefore, if in some state w we have the possibility of executing a local event, from this state there cannot be a path which always observes non-local events, since otherwise the standard model does not satisfy the divergence condition of local bisimulation.

For the other direction, suppose that M satisfies the axioms; we build a model which is standard and which is bisimilar to the original model. First, we define the following collections of states:

$$[\epsilon] \stackrel{\text{def}}{=} \{v \mid w_0 \stackrel{\epsilon}{\Rightarrow} v\} \cup \{w_0\}, \text{ and: } [es\,.\,e] \stackrel{\text{def}}{=} \{v \mid \exists z, v' : (z \in [es]) \wedge (z \stackrel{e}{\Rightarrow} v') \wedge ((v' \stackrel{\epsilon}{\Rightarrow} v) \vee v = v')\}.$$

Then we define the components of the new model as follows:

$\mathcal{W}^{\#} \stackrel{\text{def}}{=} \{e_1 \,.\, e_n \mid [e_1 \,.\, e_n] \neq \emptyset\}$.
$\mathcal{R}^{\#} \stackrel{\text{def}}{=} \{es \xrightarrow{e} es \,.\, e \mid es, es \,.\, e \in \mathcal{W}^{\#}\}$.
$\mathcal{P}^{i\#} \stackrel{\text{def}}{=} \{\langle es, e\rangle \mid \exists v \in [es] : \langle v, e\rangle \in \mathcal{P}^i|_\tau\}$.
$\mathcal{I}^{\#}(a_i) \stackrel{\text{def}}{=} \mathcal{I}|_\tau(a_i)$.
$\mathcal{I}^{\#}(p_i) \stackrel{\text{def}}{=} \{es \in \mathcal{W}^{\#} \mid \exists w \in [es] : w \in \mathcal{I}|_\tau(p_i)\}$.

Note that, if $w \in \mathcal{I}|_\tau(p)$ and $w \in [es]$, then, for every $v \in [es]$, we have $v \in \mathcal{I}|_\tau(p)$. This is because non-local events preserve propositions. This structure is well-defined since it satisfies **I1** and by definition the transitions are deterministic with respect to a given event. It is straightforward to see that this structure is standard since there are no external events. Now, we define a relationship Z as follows: $wZ[es] \Leftrightarrow w \in [es]$. Let us prove that it is a local bisimulation.

Suppose that $wZ[es]$; if $w \stackrel{\infty}{\Rightarrow}$, then we know that $[es]$ cannot diverge by non-local events. The only possibility is that there is no e such that $[es] \xrightarrow{e} [es \,.\, e]$; if there is such a transition, then w cannot diverge by non-local events, since any path which passes through w will not satisfy the axioms in $Loc(\tau)$, and then $[es]$ has no successors. Now suppose that $w \xrightarrow{e} w'$. If e is non-local, then we know that $w, w' \in [es]$, and therefore $w'Z[es]$, and we know by Property 4.4 that $L(w) = L(w') = L([es])$. If $w \xrightarrow{e} w'$ and e is local, then we know that $w' \in [es \,.\, e]$ and then $w'Z[es \,.\, e]$, and furthermore $[es] \xrightarrow{e} [es \,.\, e]$, by definition. Now, if $[es]Z^{\smile}w$, it is worth noting that $M^{\#}$ does not have any divergence via non-local events. If $[es] \xrightarrow{e} [es.e]$, we have some $w' \in [es \,.\, e]$ (by definition), and note that we have some $v \in [es]$ and $w'' \in [es \,.\, e]$ such that $v \stackrel{e}{\Rightarrow} w''$ and $w'' \stackrel{\epsilon}{\Rightarrow} w'$. Since w and v belong to $[es]$, both satisfy the same properties of L (which can be proved by a straightforward proof by induction) and therefore, since we have $v \stackrel{e}{\Rightarrow} w''$ by the axiomatic schema, we must have $v, M \vDash \langle\gamma\rangle\top$, where $I(\gamma) = e$, and therefore we have $w, M \vDash \langle\gamma\rangle\top$, i.e., there is a state $v' \in [es \,.\, e]$ such that $w \xrightarrow{e} v''$, which finishes the proof. □

Proof of Theorem 4.7. Note that, if we only take into account standard structures, the definition of $\vDash$ coincides with the definition given in [7]. Therefore, axioms **A1**–**A17** are sound with respect to standard models and then, by Theorem 4.4, these axioms are sound with respect to locus models; the same is true for axioms **TempAx1**–**TempAx5** and the rules; the rest of the axioms are straightforward using the definition of B and the relativized Done(). □

Proof of Theorem 4.8. The left direction is trivial.

For the other direction, we prove the result by induction on the length of the proof.

Base Case. If the proof is of length 1, then $\psi \in S$, or ψ is an axiom. In both cases we obtain $\vdash^L_S \mathsf{AG}\varphi \to \psi$. (If $\psi = \varphi$ it is direct to prove this sentence from the axiomatic system.)

Ind. Case. If the proof is of length greater than or equal to 1, then ψ was obtained by one of the following rules:

1. Via *modus ponens* from a formula $\varphi' \to \psi$ which appears before.
2. Via application of generalization to a formulae which appears before.
3. By induction.
4. ψ is some axiom or it belongs to S

The last case is straightforward. The other cases are dealt with as follows:

Case 1: If we obtain it by modus ponens, then $\vdash^L_S \mathsf{AG}\varphi \to \varphi'$, and $\vdash^L_S \mathsf{AG}\varphi \to (\varphi' \to \psi)$ (by induction), which using propositional logic gives us $\vdash^L_S (\mathsf{AG}\varphi \to \varphi') \to (\mathsf{AG}\varphi \to \psi)$, and using modus ponens we get $\vdash^L_S \mathsf{AG}\varphi \to \psi$.

Case 2: If we obtain ψ by generalization, then $\phi = \mathsf{AG}\psi'$. Then, we have by induction that $\vdash^L_S \mathsf{AG}\varphi \to \psi'$; applying generalization we get $\vdash^L_S \mathsf{AG}(\mathsf{AG}\varphi \to \psi')$, and then it is straightforward using the axioms to prove $\vdash^L_S \mathsf{AGAG}\varphi \to \mathsf{AG}\psi'$. But we have that $\vdash^L_S \mathsf{AGAG}\varphi \leftrightarrow \mathsf{AG}\varphi$, then using this property we have $\vdash^L_S \mathsf{AG}\varphi \to \mathsf{AG}\psi'$. For modal generalization the proof is similar.

Case 3: If we obtained ψ by induction, this means that $\vdash^L_{S,\varphi} \mathsf{B} \to \psi$ and $\vdash^L_{S,\varphi} [\mathbf{U}]\psi$, and then by induction we obtain $\vdash^L_S \mathsf{AG}\varphi \to (\mathsf{B} \to \psi)$ and $\vdash^L_S \mathsf{AG}\varphi \to [\mathbf{U}]\psi$, and then we have that $\mathsf{AG}\varphi \vdash^L_S \mathsf{B} \to \psi$ and $\mathsf{AG}\varphi \vdash^L_S [\mathbf{U}]\psi$. But, then, using the induction rule we get $\mathsf{AG}\varphi \vdash^L_S \psi$ and then using the deduction theorem for the local notion of deduction we get $\vdash^L_S \mathsf{AG}\varphi \to \psi$. □

Proof of Theorem 4.9. We prove that the translation of every axiom of the deductive system of L is a theorem of the deductive system of L', and for the deduction rules, if we have the translation of the premises, then we can prove the conclusion, and therefore every proof in L can be simulated in L', modulo translation.

For the axioms of the propositional part of the logic, only two axioms are dependent on the language: A12 and A17. For A17, note that the translation of the instances of this axiom, $\tau(\langle\gamma\rangle\varphi \to [\gamma]\varphi)$, is exactly the axioms of atomicity, and therefore: $\vdash^{L'}_{Loc(\tau)} \tau(\langle\gamma\rangle\varphi \to [\gamma]\varphi)$. For the axiom A12, the proof is similar. And since the other axioms are not dependent on the language, the translation of these axioms are instances of axioms in L'. For the deduction rules of the propositional

part (modus ponens and modal generalization) the proof is straightforward. For the temporal axioms and rules we proceed by cases.

TempAx1: We need to prove: $\vdash^{L'}_{Loc(\tau)} \tau(\langle \mathbf{U}\rangle \top \rightarrow \mathsf{A}\varphi \leftrightarrow [\mathbf{U}]\varphi)$. Note the following property of Done(): $\langle\alpha\rangle\top \rightarrow ((\mathsf{AN}\mathsf{Done}(\alpha) \rightarrow \varphi) \leftrightarrow [\alpha]\varphi)$. Using this property, we obtain that the sentence above is equivalent to: $\langle\tau(\mathbf{U})\rangle\top \rightarrow ([\tau(\mathbf{U})]\tau(\varphi) \leftrightarrow [\tau(\mathbf{U})]\tau(\varphi))$ which is obviously a theorem of $\vdash^{L'}_{Loc(\tau)}$.

TempAx2: It is straightforward that $[\tau(\mathbf{U})]\bot \vdash^{L'}_{Loc(\tau)} \tau(\varphi) \leftrightarrow \tau(\varphi)$ and the property follows.

TempAx3: The translation of this axiom is an instance of the same axiom in L'.

TempAx4: Proving the right direction of the implication is direct; let us prove:

$$\tau(\psi) \vee \tau(\mathsf{ENE}(\varphi\ \mathcal{U}\ \psi)) \vdash^{L'}_{Loc(\tau)} \mathsf{E}(\varphi\ \mathcal{U}\ \psi)$$

Using the property of Done() described above; we obtain that the left part of the assertion above is equivalent to: $\tau(\psi) \vee (\tau(\varphi) \wedge (\langle\tau(\varphi)\rangle\top \rightarrow \langle\tau(\varphi)\rangle\mathsf{E}(\tau(\varphi)\ \mathcal{U}\ \tau(\psi)))) \wedge ([\tau(\alpha)]\varphi \rightarrow \tau(\varphi))$. Simple calculations (using the axioms for AN) show that from this formula we can prove $\mathsf{E}(\tau(\varphi)\ \mathcal{U}\ \tau(\psi))$.

TempAx5: We have to prove: $\tau(\psi) \wedge (\tau(\varphi) \wedge \tau(\mathsf{ANA}(\varphi\ \mathcal{U}\ \psi))) \vdash^{L'}_{Loc(\tau)} \mathsf{A}(\varphi\ \mathcal{U}\ \psi)$ Using the definition of τ and properties of Done(), the left part is equivalent to:

$$\tau(\psi) \vee (\tau(\varphi) \wedge (\langle\tau(\mathbf{U})\rangle\top \rightarrow [\tau(\mathbf{U})]\mathsf{A}(\tau(\varphi)\ \mathcal{U}\ \tau(\psi))) \wedge [\tau(\mathbf{U})]\bot \rightarrow \tau(\psi)). \tag{A.1}$$

Note that by locality we have that: $\vdash^{L'}_{Loc(\tau)} \varphi \rightarrow \mathsf{A}(\varphi\mathsf{WDone}(\tau(\mathbf{U})))$, and note that:

$$(\varphi \rightarrow \mathsf{A}(\varphi\mathsf{WDone}(\tau(\mathbf{U})))) \wedge \mathsf{AFDone}(()\tau(\mathbf{U})) \vdash^{L'}_{Loc(\tau)} \varphi \rightarrow \mathsf{A}(\varphi\ \mathcal{U}\ \mathsf{Done}(\tau(\mathbf{U}))).$$

Using the fact that we have the following axiomatic schema in $Loc(\tau)$: $\langle\tau(\mathbf{U})\rangle\top \rightarrow \mathsf{AFDone}(\tau(\mathbf{U}))$, and that from the formula (A.1) we obtain $\mathsf{Done}(\tau(\mathbf{U})) \rightarrow \mathsf{A}(\tau(\varphi)\ \mathcal{U}\ \tau(\psi))$, we have that:

$$\varphi \wedge \langle\tau(\mathbf{U})\rangle\top \vdash^{L'}_{Loc(\tau)} \mathsf{A}(\tau(\varphi)\ \mathcal{U}\ (\mathsf{A}(\tau(\varphi)\ \mathcal{U}\ \tau(\psi))))$$

which is equivalent to: $\varphi \wedge \langle\tau(\mathbf{U})\rangle\top \vdash^{L'}_{Loc(\tau)} \mathsf{A}(\tau(\varphi)\ \mathcal{U}\ \tau(\psi))$. The result follows.

Axioms **TempAx6**–**TempAx9** are straightforward as their translations are instances of the same axioms.

TempAx8: The translation of this axiom is $[\tau(\mathbf{U})]\neg\mathsf{B}$, which can be proven using the properties of modalities.

TempAx11 and **TempAx12** are direct.

For the induction rule we can proceed as follows. Note that, if we have $\vdash^{L'}_{Loc(\tau)} \mathsf{B} \rightarrow \tau(\varphi)$, and $\vdash^{L'}_{Loc(\tau)} \tau(\varphi) \rightarrow [\tau(\mathbf{U})]\tau(\varphi)$, then we have: $\vdash^{L'}_{Loc(\tau)} \tau(\varphi) \rightarrow [\tau(a_1)\sqcup\cdots\sqcup\tau(a_n)]\tau(\varphi)$, and by locality we have: $\vdash^{L'}_{Loc(\tau)} \tau(\varphi) \rightarrow [\overline{\tau(a_1)\sqcup\cdots\sqcup\tau(a_n)}]\tau(\varphi)$, and then using the properties of the logic we get: $\vdash^{L'}_{Loc(\tau)} \tau(\varphi) \rightarrow [\mathbf{U}]\tau(\varphi)$, and then using the induction rule we get: $\vdash^{L'}_{Loc(\tau)} \tau(\varphi)$.

The temporal rule **TempRule2** is straightforward. For the rule **TempRule3**, if we have:

$$\vdash^{L'}_{Loc(\tau)} \tau(\varphi) \rightarrow (\neg\tau(\psi) \wedge \tau(\varphi))$$

this is equivalent to: $\vdash^{L'}_{Loc(\tau)} \tau(\varphi) \rightarrow \neg\tau(\psi) \wedge (\langle\tau(\mathbf{U})\rangle\tau(\varphi) \vee [\tau(\mathbf{U})]\bot \rightarrow \tau(\varphi))$. It is not hard to prove that this formula implies $\tau(\varphi) \rightarrow (\neg\tau(\psi) \vee \mathsf{EN}\tau(\varphi))$, and then, applying **TempRule3**, we obtain $\neg\mathsf{A}(\tau(\varphi)\ \mathcal{U}\ \tau(\psi))$. For the rule **TempRule4**, the proof is similar using the locality axioms. □

Proof of Theorem 5.2. The identity arrow is the identity translation, which obviously satisfies all the requisites. And the composition between mappings is just the composition of the functions which define these mappings. In addition, we must prove that $\vdash_C Loc(\mathrm{id}_C)$ (where id_C is the identity translation). And, if we have translations $\tau : C_1 \rightarrow C_2$ and $\tau' : C_2 \rightarrow C_3$, then $\vdash_{C_3} Loc(\tau' \circ \tau)$.

To prove $\vdash_C Loc(id)$, we have to prove (i) $\vdash_C \langle\gamma\rangle\top \rightarrow \langle\gamma\rangle\top$, (ii) $\vdash_C \varphi \rightarrow [\overline{\mathbf{U}}]\varphi$, (iii) $\langle\mathbf{U}\rangle\top \rightarrow \mathsf{AFDone}(\mathbf{U})$ and (iv) $\vdash_C \langle\gamma\rangle\varphi \rightarrow [\gamma]\varphi$. (i), (ii) and (iv) are straightforward from the axioms. For (iii) we have that $\vdash_C \langle\mathbf{U}\rangle\top \rightarrow \langle\mathbf{U}\rangle\mathsf{Done}(\mathbf{U})$ by definition of Done(), and also $\vdash_C [\mathbf{U}]\mathsf{Done}(\mathbf{U})$, by the temporal axioms we have $\vdash_C \mathsf{AN}\varphi \wedge \mathsf{EN}\varphi \rightarrow \mathsf{AF}\varphi$, and then using **TempAx** we get $\vdash_C \langle\mathbf{U}\rangle\top \rightarrow \mathsf{AFDone}(\mathbf{U})$.

Now, we have to prove $\vdash_{(C_3)} Loc(\tau' \circ \tau)$. We have that: $\vdash_{C_2} \langle\tau(\gamma)\rangle\top \rightarrow \langle\tau(\gamma) \sqcap a_1 \sqcap \cdots \sqcap a_n\rangle\top$ (where $a_1, \ldots, a_n$ are the primitive action which are not images of any symbol by τ). Therefore, by properties of translations we have, $\vdash_{C_3} \langle(\tau' \circ \tau)(\gamma)\rangle\top \rightarrow \langle\rangle\tau'(\overline{a_1}) \sqcap \cdots \sqcap \tau'(\overline{a_n})\rangle\top$, where $b_1, \ldots, b_n$ are the primitive action of Φ^3_0 which are not in the image of τ'. Since $[(\tau' \circ \tau)(\gamma) \sqcap \tau'(\overline{a_1}) \sqcap \cdots \sqcap \tau'(\overline{a_n})]$ is an atomic action term in the language of C_2, we have that:

$$\vdash_{C_3} \langle(\tau' \circ \tau)(\gamma) \sqcap \tau'(\overline{a_1}) \sqcap \cdots \sqcap \tau'(\overline{a_n})\rangle\top \rightarrow \langle(\tau' \circ \tau)(\gamma) \sqcap \tau' \circ \tau(\overline{a_1}) \sqcap \cdots \sqcap \tau' \circ \tau(\overline{a_n}) \sqcap b_1 \sqcap \cdots \sqcap b_m\rangle\top$$

let $b'_1, \ldots, b'_k$ be the primitive actions in the language of C_3 which are not translations of any primitive action of C through $\tau' \circ \tau$. Note that, if some of these b'_j is a translation of a primitive action a_j of C_2, then $\vdash_{C_3} \tau'(a_j) \sqcap \overline{b'_j}$, otherwise b'_j is some of

the b_i's. In any case we have:

$$\vdash_{C_3} (\tau' \circ \tau)(\gamma) \sqcap \tau' \circ \tau(\overline{a_1}) \sqcap \cdots \sqcap \tau' \circ \tau(\overline{a_n}) \sqcap b_1 \sqcap \cdots \sqcap b_m \sqsubseteq (\tau' \circ \tau)(\gamma) \sqcap b'_1 \sqcap \cdots \sqcap b'_k \sqcap b_1 \sqcap \cdots \sqcap b_m.$$

Therefore we have: $\vdash_{C_3} \langle \tau' \circ \tau(\gamma)\rangle \top \rightarrow \langle (\tau' \circ \tau)(\gamma) \sqcap b'_1 \sqcap \cdots \sqcap b'_k \sqcap b_1 \sqcap \cdots \sqcap b_m\rangle \top$.

On the other hand, we have $\vdash_{C_2} \tau(\varphi) \rightarrow [\tau(\overline{\mathbf{U}})]\tau(\varphi)$, and properties of translation we have: $\vdash_{C_3} \tau' \circ \tau(\varphi) \rightarrow [\tau' \circ \tau(\overline{\mathbf{U}})]\tau' \circ \tau(\varphi)$. We also have: $\vdash_{C_2} \langle \tau(\gamma)\rangle \tau(\varphi) \rightarrow [\tau(\gamma)]\tau(\varphi)$, and by Theorem 4.5 we have $\vdash_{C_3} \langle \tau' \circ \tau(\gamma)\rangle \tau' \circ \tau(\gamma) \rightarrow [\tau' \circ \tau(\gamma)]\tau' \circ \tau(\gamma)$. The same reasoning can be used to prove: $\vdash_{C_3} \langle \tau' \circ \tau(\mathbf{U})\rangle \top \rightarrow \mathsf{AF}(\tau' \circ \tau(\mathsf{Done}(\mathbf{U})))$. □

Proof of Theorem 5.3. We prove that the functor $Sign$: **Comp** $\rightarrow$ **Sign** reflects colimits, and since **Sign** is finitely cocomplete, hence **Comp** is finitely cocomplete too.

Suppose that $D : I \rightarrow$ **Comp** is a diagram in **Comp**. Therefore, we have a diagram $D' = SignD : I \rightarrow$ **Sign**. Say $C_i = \langle L_i, A_i\rangle$ are the components of the diagram. Let $\langle L, \alpha : D' \rightarrow L\rangle$ be a colimit cocone in **Sign**; then we assert that

$$C = \left\langle L, \bigcup_{i\in I} \alpha_i(A_i) \cup \bigcup_{i\in I} \alpha_i(\mathrm{Loc}(C_i)) \right\rangle$$

is a colimit object in **Comp**. For each component C_i, the translation to C is given by α_i. We prove that α_i is a morphism between components. We know that $\vdash^L_{\bigcup_{i\in I}\alpha_i(A_i)} \alpha_i(A_i)$ and $\vdash^L_{\bigcup_{i\in I}\alpha_i(\mathrm{Loc}(C_i))} \alpha_i(\mathrm{Loc}(C_i))$ and $\vdash^L_{\bigcup_{i\in I}\alpha_i(G(L_i))} \alpha_i(G(L_i))$, and by Theorem 5.1 we have that $\vdash_\Gamma \alpha_i(\varphi)$, where $\Gamma = \bigcup_{i\in I} \alpha_i(A_i) \cup \bigcup_{i\in I} \alpha_i(\mathrm{Loc}(C_i))$, for every $\vdash_{C_i} \varphi$, and therefore α_i is a morphism between components. These morphisms make the corresponding diagram commute in **Sign**, and therefore their extension make the corresponding diagram commute in **Comp**. Now, if we have another cocone $\langle C', \beta : C_i \rightarrow C'\rangle$, then in **Sign** we have an unique morphism $\psi : L \rightarrow L'$ (where L' is the language of C'). It is straightforward to check that ψ can be extended to an unique $\psi : C \rightarrow C'$, extending the mapping of languages to mapping between formulae. This finishes the proof. □

Proof of Theorem 6.2. Suppose that we have an upgrading transition $V \rightarrow V'$ in $U_{C_1} + U_{C_2}$. For the case that $V \rightarrow V'$ belongs to U_{C_1}, we proceed as follows: let $a_1, \ldots, a_k$ be the primitive actions of C_1 and $b_1, \ldots, b_m$ be the primitive actions of C_2. Let us use $C_1 - C_2$ for the expression:

$$\bigsqcup_{\tau_2(b_i)\notin\{\tau_1(a_1),\ldots,\tau_1(a_n)\}} \tau_2(b_i).$$

and similarly for $C_2 - C_1$. We have that: $\vdash_{C_1+_C C_2} \tau_1(V) \rightarrow [\tau_1(\alpha_1); \ldots; \tau_1(\alpha_n)]\tau(V') \wedge \langle \tau_1(\alpha_1); \ldots; \tau_1(\alpha_n)\rangle \top$. Note that $\tau_1(V)$ and $\tau_1(V')$ are not necessarily violation states of $C_1 +_C C_2$. Now let $\mathtt{v}_1, \ldots, \mathtt{v}_t$ be the violation predicates which do not appear in $\tau_1(V)$. Obviously, these violation predicates are translations of violation predicates of component C_2. Now by locality we have:

$$\vdash_{C_1+_C C_2} \neg\mathtt{v}_1 \wedge \cdots \neg\mathtt{v}_t \rightarrow [\overline{\tau_2(\mathbf{U})}]\neg\mathtt{v}_1 \wedge \cdots \wedge \neg\mathtt{v}_t.$$

Also we know that:

$$\vdash_{C_1+_C C_2} \neg\mathtt{v}_1 \wedge \cdots \neg\mathtt{v}_t \rightarrow [c_1 \sqcup \cdots \sqcup c_n]\neg\mathtt{v}_1 \wedge \cdots \wedge \neg\mathtt{v}_t.$$

since $c_1, \ldots, c_n$ are safe actions by hypothesis and by the translations of the axioms $SG(C_2)$ and therefore $\mathbf{P}(c_i)$ for any i. Now using the formulae above and the properties of the logic we get:

$$\vdash_{C_1+_C C_2} \neg\mathtt{v}_1 \wedge \cdots \neg\mathtt{v}_t \rightarrow [(c_1 \sqcup \cdots \sqcup c_n) \sqcup \overline{\tau_2(\mathbf{U})}]\neg\mathtt{v}_1 \wedge \cdots \wedge \neg\mathtt{v}_t$$

and $(c_1 \sqcup \cdots \sqcup c_n) \sqcup \overline{\tau_2(\mathbf{U})}$ is just $\overline{C_2 - C_1}$.

$$\vdash_{C_1+_C C_2} \tau_1(V) \wedge \neg\mathtt{v}_1 \wedge \cdots \wedge \neg\mathtt{v}_t \rightarrow [\alpha_1 \sqcap \overline{C_2 - C_1}; \ldots; \alpha_n \sqcap \overline{C_2 - C_1}]\tau_1(V') \wedge \neg\mathtt{v}_1 \wedge \cdots \wedge \neg\mathtt{v}_t.$$

We find here part of the formula that we must prove. For the other part we have:

$$\vdash_{C_1+_C C_2} \tau_1(V) \rightarrow \langle \alpha_1; \ldots; \alpha_n\rangle \top.$$

Consider that $C_2 - C_1$ are exactly the choice of the action which belongs to $C_1 +_C C_2$ and do not belong to the translation of primitive actions in C_1, and therefore by independence we get:

$$\vdash_{C_1+_C C_2} \tau_1(V) \rightarrow \langle \alpha_1 \sqcap \overline{C_2 - C_1}; \ldots; \alpha_n \sqcap \overline{C_2 - C_1}\rangle \top.$$

The case that $V' \rightarrow V$ in U_{C_2} uses a similar argument. This finishes the proof. □

Proof of Theorem 6.4. Suppose that we have an arrow $V \to V'$ in $U_{C_1} + U_{C_2}$, then let us suppose that $V \to V'$ belongs to U_{C_1} (the proof for the other case is similar). Let C denote the tip of the colimit of the diagram: $C_1 \leftarrow p_1 \to I \leftarrow p_2 \to C_2$, and let $\tau_1 : C_1 \to C$, $\tau_2 : C_2 \to C$ and $\tau_I : I \to C$ be the arrow from the base of the colimit to the tip. Since we have the arrow $V \to V'$ in the upgrading diagram of C_1, we have: $\vdash_{C_1} V \to [\alpha_1; \ldots; \alpha_n]V \wedge V \to \langle\alpha_1; \ldots; \alpha_n\rangle\top$. By the property of mappings between components we have that: $\vdash_{C_1} \tau_1(V) \to [\tau_1(\alpha_1); \ldots; \tau_1(\alpha_n)]V \wedge V \to \langle\tau_1(\alpha_1); \ldots; \tau_1(\alpha_n)\rangle\top$. Now, $\tau_1(V)$ and $\tau_1(V')$ are not violation states of C, since some violations (say $\mathrm{v}_1, \ldots, \mathrm{v}_k$) do not appear in these predicates. These predicates are translations of the violation in the language of C_2. By the axioms of locality we get: $\vdash_C \neg\mathrm{v}_1 \wedge \cdots \wedge \neg\mathrm{v}_k \to [\tau_1(\alpha_1) \sqcap \overline{\tau_2(\mathbf{U})}; \ldots; \tau_1(\alpha_n) \sqcap \overline{\tau_2(\mathbf{U})}]\neg\mathrm{v}_1 \wedge \cdots \wedge \neg\mathrm{v}_n$, and therefore we have:

$$\vdash_{C_1+_I C_2} \tau_1(V) \wedge \neg\mathrm{v}_1 \wedge \cdots \wedge \neg\mathrm{v}_k \to [\tau_1(\alpha_1) \sqcap \overline{\tau_2(\mathbf{U})}; \ldots; \tau_1(\alpha_n) \sqcap \overline{\tau_2(\mathbf{U})}]\neg\mathrm{v}_1 \wedge \cdots \wedge \neg\mathrm{v}_n \wedge \tau_1(V'). \tag{A.2}$$

On the other hand, consider the actions $a_1, \ldots, a_n$ that are translations of actions in C_2; for any i, if a_i is not the translation of an action in C_1, then by independence we get: $V \to \langle\tau_1(\alpha_1) \sqcap \overline{\tau_2(a_i)}; \ldots; \tau_1(\alpha_n) \sqcap \overline{\tau_2(a_i)}\rangle\top$, otherwise, we have that (by hypothesis) $[a_i]\bot$, so $\tau_1(V) \to \langle\tau_1(\alpha_1); \ldots; \tau_1(\alpha_n)\rangle\tau_1(V')$ implies $\tau_1(V) \to \langle\tau_1(\alpha_1) \sqcap \overline{\tau_2(a_i)}; \ldots; \tau_1(\alpha_n) \sqcap \overline{\tau_2(a_i)}\rangle\tau_1(V')$, that is, in either case we obtain:

$$\vdash_{C_1+_I C_2} \tau_1(V) \to \langle\tau_1(\alpha_1) \sqcap (\overline{a_1} \sqcup \cdots \sqcup \overline{a_n}); \ldots; \tau_1(\alpha_n) \sqcap (\overline{a_1} \sqcup \cdots \sqcup \overline{a_n})\rangle\tau_1(V') \tag{A.3}$$

which is equivalent to $\tau_1(V) \to \langle\tau_1(\alpha_1) \sqcap \overline{\tau_2(\mathbf{U})}; \ldots; \tau_1(\alpha_n) \sqcap \overline{\tau_2(\mathbf{U})}\rangle\tau_1(V')$. Putting together Eqs. (A.2) and (A.3) we get:

$$\vdash_{C_1+_I C_2} (\tau_1(V) \wedge \neg\mathrm{v}_1 \wedge \cdots \wedge \neg\mathrm{v}_k \to [\tau_1(\alpha_1) \sqcap \overline{\tau_2(\mathbf{U})}; \ldots; \tau_1(\alpha_n) \sqcap \overline{\tau_2(\mathbf{U})}]\tau_1(V') \wedge \neg\mathrm{v}_1 \wedge \cdots \wedge \neg\mathrm{v}_k) \wedge$$
$$(\tau_1(V) \wedge \neg\mathrm{v}_1 \wedge \cdots \wedge \neg\mathrm{v}_k \to \langle\tau_1(\alpha_1) \sqcap \overline{\tau_2(\mathbf{U})}; \ldots; \tau_1(\alpha_n) \sqcap \overline{\tau_2(\mathbf{U})}\rangle\top)$$

this finishes the proof. □

References

[1] A. Pnueli, The temporal logic of programs, in: Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, 1977, pp. 46–67.
[2] M. Ben-Ari, Z. Manna, A. Pnueli, The temporal logic of branching time, Acta Inform. 20 (1983) 207–226.
[3] E. Emerson, Temporal and Modal Logic, in: Handbook of Theorical Computer Science, Elsevier, 1995, pp. 995–1072.
[4] L. Aqvist, Deontic logic, in: D.M. Gabbay, F. Guenther (Eds.), Handbook of Philosophical Logic, vol. 2, Kluwer Academic Publishers, 1984, pp. 605–714.
[5] R. J. Wieringa, J.-J. Meyer, Applications of deontic logic in computer science: a concise overview, Deontic Logic in Computer Science, Normative System Specification.
[6] S. Khosla, T. Maibaum, The prescription and description of state-based systems, in: H. B. Banieqnal, A. Pnueli (Eds.), Temporal Logic in Computation, Springer-Verlag, 1985.
[7] P. F. Castro, T. Maibaum, Deontic action logic, atomic boolean algebra and fault-tolerance, J. Appl. Log. 7 (4) (2009) 441–466.
[8] P. F. Castro, T. Maibaum, Characterizing locality (encapsulation) with bisimulation, in: Theoretical Aspects of Computing — ICTAC, 2010.
[9] J.L. Fiadeiro, T. Maibaum, Towards object calculi, in: S.A. Saake G (Ed.) Information Systems; Correctness and Reusability, Technische Universität Braunschweig, 1991.
[10] D. L. Parnas, On the criteria to be used in decomposing systems into modules, Commun. ACM 15 (12) (1972) 1053–1058.
[11] R. Burstall, J. Goguen, Putting theories together to make specifications, in: R.Reddy (Ed.), Porc. Fifth International Joint Conference on Artificial Intelligence, 1977.
[12] S. MacLane, Categories for the Working Mathematician, Springer-Verlag, 1998.
[13] J. L. Fiadeiro, T. Maibaum, Temporal theories as modularization units for concurrent system specification, in: Formal Aspects of Computing, vol. 4, 1992, pp. 239–272.
[14] M. J. Sergot, H. Prakken, Contrary-to-duty obligations, in: DEON 94 (Proc.Second International Workshop on Deontic Logic in Computer Science), 1994.
[15] H. Barringer, The use of temporal logic in the compositional specification of concurrent systems, in: A. Galton (Ed.), Temporal Logic and Their Applications, Academic Press, 1987.
[16] G. Gargov, S. Passy, A note on boolean logic, in: P.P. Petkov (Ed.), Proceedings of the Heyting Summerschool, Plenum Press, 1990.
[17] P. Blackburn, M. Rijke, Y. Venema, Modal Logic, in: Cambridge Tracts in Theoretical Computer Science, vol. 53, 2001.
[18] C. Lutz, U. Sattler, The complexity of reasoning with boolean modal logics, in: Advances in Modal Logic 3, World Scientific, 2000, pp. 329–348.
[19] P. F. Castro, T. Maibaum, A tableaux system for deontic action logic, in: Proceedings of 9th International Conference on Deontic Logic in Computer Science, Springer-Verlag, Luxembourg, 2008.
[20] J. Broersen, Modal action logics for reasoning about reactive systems, Ph.D. Thesis, Vrije University (2003).
[21] P. F. Castro, Deontic action logics for the specification and analysis of fault-tolerance, Ph.D. Thesis, McMaster University, Department of Computing and Software, 2009.
[22] J. Meyer, A different approach to deontic logic: deontic logic viewed as variant of dynamic logic, in: Notre Dame Journal of Formal Logic, vol. 29, 1988.
[23] F. Dignum, R. Kuiper, Combining dynamic deontic logic and temporal logic for the specification of deadlines, in: Proceedings of the thirtieth HICSS, 1997.
[24] S. Kent, B. Quirk, T. Maibaum, Specifying deontic behaviour in modal action logic, Tech. rep., Forest Research Project, 1991.
[25] R. DeNicola, F. Vaandrager, Three logics for branching bisimulation, J. ACM 42 (1995) 458–487.
[26] L. Lamport, Specifying concurrent program modules, ACM Trans. Program. Lang. Syst. 5 (1983) 190–222.
[27] L. Lamport, A simple approach to specifying concurrent systems, Commun. ACM 32 (1) (1989) 32–45.
[28] J. L. Fiadeiro, T. S. E. Maibaum, Sometimes "tomorrow" is "sometime" — action refinement in a temporal logic of objects, in: ICTL, 1994, pp. 48–66.
[29] J. Goguen, R. Burstall, Institutions: abstract model theory for specification and programming, in: Journal of the Association of Computing Machinery, 1992.
[30] J. L. Fiadeiro, A. Sernadas, Structuring theories on consequence, in: Recent Trends in Data Type Specification, 5th Workshop on Abstract Data Types, Gullane, Scotland, Selected Papers, 1987, pp. 44–72.
[31] J. Adámek, H. Herrlich, G. Strecker, Abstract and Concrete Categories: The Joy of Cats, John Wiley and Sons, 2009, corrected version of the 1990 book of the same name, available online at http://katmat.math.uni-bremen.de/acc/.
[32] S. MacLane, I. Moerdijk, Sheaves in Geometry and Logic, Springer-Verlag, 1992.
[33] M. J. Sergot, R. Craven, The deontic component of action language nC+, DEON (2006) 222–237.
[34] J. Fiadeiro, V. Schmitt, Structured co-spans: an algebra of interaction protocols, in: Algebra and Coalgebra in Computer Science, Second International Conference, CALCO 2007, Bergen, Norway, 2007.

[35] J.-R. Abrial, The B-Book: Assigning Programs to Meanings, Cambridge University Press, 1996.
[36] G. Smith, The Object-Z Specification Language, Kluwer Academic Publishers, 2000.
[37] Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems: Specification, Springer, 1992.
[38] M. Abadi, L. Lamport, Conjoining specifications, ACM Trans. Program. Lang. Syst. 17 (1995) 507–534.
[39] C. Baier, J.-P. Katoen, Principles of Model Checking, MIT Press, 2008.
[40] J. Fiadeiro, T. S. E. Maibaum, Interconnecting formalisms: supporting modularity, reuse and incrementality, in: SIGSOFT FSE, 1995, pp. 72–80.
[41] M. Abadi, L. Lamport, Composing specifications, ACM Trans. Program. Lang. Syst. 15 (1) (1993) 73–132.
[42] L. de Alfaro, T. A. Henzinger, Interface automata, in: ESEC / SIGSOFT FSE, 2001.
[43] L. de Alfaro, T. A. Henzinger, Interface theories for component-based design, in: Lecture Notes in Computer Science, vol. 2211, Springer, 2001, pp. 148–165.
[44] J. L. Fiadeiro, Categories for Software Engineering, Springer-Verlag, 2005.
[45] H. E. Enderton, A Mathematical Introduction to Logic, Academic Press, 1972.
[46] M. Hennessy, R. Milner, On observing nondeterminism and concurrency, ICALP (1980) 299–309.
[47] R. J. van Glabbeek, W. P. Weijland, Branching time and abstraction in bisimulation semantics, J. ACM 43 (3) (1996) 555–600.
[48] M. C. Browne, E. M. Clarke, O. Grumberg, Characterizing kripke structures in temporal logic, APSOFT'87: Proceedings of the International Joint Conference on Theory and Practice of Software Development, Pisa, Italy.
[49] F. Cristian, A rigorous approach to fault-tolerant programming, IEEE Trans. Softw. Eng. 11 (1985) 23–31.
[50] H. Schepers, R. Gerth, A compositional proof theory for fault tolerant real-time distributed systems, in: SRDS, 1993, pp. 34–43.
[51] I. S.W. B. Prasetya, S. D. Swierstra, Formal design of self-stabilizing programs, J. High Speed Netw. 14 (2005) 59–83.
[52] A. Arora, A foundation of fault-tolerant computing, Ph.D. Thesis, The University of Texas at Austin, 1992.
[53] D. Peled, M. Joseph, A compositional framework for fault tolerance by specification transformation, Theor. Comput. Sci. 128 (1994) 99–125.
[54] R. M. Amadio, S. Prasad, Localities and failures, in: Foundations of Software Technology and Theoretical Computer Science, 14th Conference, Madras, India, 1994.
[55] T. Janowski, Bisimulation and fault-tolerance, Ph.D. thesis, Department of Computer Science, University of Warwick, 1995.
[56] L. Lamport, S. Merz, Specifying and verifying fault-tolerant systems, in: Formal Techniques in Real-Time and Fault-Tolerant Systems, Third International Symposium Organized Jointly with the Working Group Provably Correct Systems - ProCoS, 1994, pp. 41–76.
[57] L. Lamport, R. E. Shostak, M. C. Pease, The byzantine generals problem, ACM Trans. Program. Lang. Syst. 4 (1982) 382–401.
[58] E. Kang, D. Jackson, Formal modeling and analysis of a flash filesystem in alloy, in: Abstract State Machines, B and Z, Springer, Berlin, Heidelberg, 2008.
[59] D. Yadav, M. Butler, Formal development of a total order broadcast for distributed transactions using Event-B, in: Methods, Models and Tools for Fault Tolerance, Springer, 2009.
[60] Z. Liu, M. Joseph, Specification and verification of fault-tolerance, timing, and scheduling, ACM Trans. Program. Lang. Syst. 21 (1) (1999) 46–89.
[61] K. G. Larsen, B. Thomsen, A modal process logic, in: LICS, 1988, pp. 203–210.