A Deontic Logic for the Support of Modularity

Pablo F.Castro
castropf@mcmaster.ca
Department of Computing & Software
McMaster University
Hamilton, Canada

T.S.E.Maibaum
tom@maibaum.org
Department of Computing & Software
McMaster University
Hamilton, Canada

Abstract

In previous work [7], we have presented a deontic logic to specify and verify fault-tolerant software. However, this logic lacks the notion of module or component. In this report we investigate formal mechanisms to allow designers to decompose a specification (stated in this logic) into several components. Furthermore, because the notion of violation or error state is very important when reasoning about fault-tolerance, we study how the decomposition of a specification into modules affects the underlying logical structure of violations.

Key words: Fault-Tolerance, Deontic Logic, Software Specification, Software Verification, Component Based Design, Category Theory.

1 Introduction

In this report we introduce some modifications to the deontic logic presented in [7] with the aim of obtaining a more general framework where system specifications can be written in a modular way. For this purpose, we mainly follow the philosophy of [4], in the sense that a system is specified by putting together smaller specifications (by means of some categorical constructions). The ideas presented below are also inspired by the logical frameworks presented in [11] and [12], where Goguen's ideas are applied to temporal logics and, therefore, to specifications of concurrent systems and object oriented systems, respectively. In [11] a logic with support for modularization, with temporal constructs, is presented. This logic incorporates deontic predicates (permission, obligation, prohibition) to give a broader language to specify objects (or modular units); deontic predicates (as shown in [6]) allow designers to separate the concepts of description and prescription of pieces of software. The descriptional part of the logic of a component describes what the effects of the actions are of this component (in a pre/post-condition style). On the other hand, the deontic aspect describes how this module **should** behave, though a component may exhibit a different behavior to that which is expected. In [11] a linear temporal logic is used and the deontic constructs are global, in the sense that the prescriptions given in terms of them are shared by all the components of the specification. Here we present a branching time logic (which reflects in some way that the notion of non-determinism is embedded inside of this logic). Moreover, we change some definitions in DPL so that the prescriptions in one component do not affect the other components in the system (i.e., they are intended to be *local*).

The report is organized as follows. In section 2 we present the logical machinery which allows us to specify different modular units of a system. In particular, we exhibit a formal framework, which is useful when analyzing the violations that may occur during the execution of the system being specified. The deontic operators play a key role here: violations arise since an obligation is not fulfilled, or a prohibition is violated. Interestingly, some natural properties of allowed actions (namely that an allowed action does not introduce more violations in a given state) are useful to facilitate the reasoning about the behavior of the system in faulty scenarios. We give some simple examples which are intended to show the application of these ideas in practice.

2 Modularizing the Deontic Logic

We introduce some changes in the vocabularies, as defined in [7], to get a more expressive language. Here we use vocabulary (or language) to refer to a tuple $L = \langle \Delta_0, \Phi_0, V_0, I_0 \rangle$, where Δ_0 (as before) is a finite set of *primitive actions*: $a_1, ..., a_n$, which represent the possible actions of a part of the system and, perhaps, of its environment. Φ_0 is an enumerable set of propositional symbols denoted by $p_1, p_2, \ldots$. V_0 is a finite subset of $\mathcal{V}$, where $\mathcal{V} = \{\mathtt{v}_1, \mathtt{v}_2, \mathtt{v}_2, \ldots\}$ is an infinite, enumerable set of "violation" propositions, and I_0 is a finite index set of (categories of) permissions. All these sets are mutually disjoint.

The set V_0 could be included in of Φ_0; however, for our purposes, it is better to have these propositional variables separated from the usual variables. This is a slight difference with the logic defined in [5], and it allows us to define translations between specifications of components in a coherent way. On the other hand, given a vocabulary V, the set of actions Δ are terms made up of primitive actions and the usual boolean connectors, denoted by: $\alpha \sqcup \beta, \alpha \sqcap \beta, \overline{\alpha}$. As before, we use Greek letters to range over actions. We consider the same formulae as in [5], but we add some novel predicates. The formal definition of the formulae is:

- If $\varphi \in \Phi_0 \cup V_0$, then $\varphi \in \Phi$.
- If φ and ψ are formulae, then $\psi \to \psi, \neg\psi \in \Phi$.
- If φ is a formula and α an action, then $[\alpha]\varphi$ is a formula.
- If α is an action and $i \in I_0$, then $\mathsf{P}^i_{\mathsf{w}}(\alpha)$ and $\mathsf{P}^i(\alpha)$ are formulae.
- If φ and ψ are formulae, then $\mathsf{EN}\varphi$, $\mathsf{A}(\varphi\,\mathcal{U}\,\psi)$ and $\mathsf{E}(\varphi\,\mathcal{U}\,\psi) \in \Phi$.

We also define a modified version of the $\mathsf{Done}()$ operator; this operator can be thought of as being a restriction of the standard $\mathsf{Done}()$ operator relativised to a restricted set of actions.

- If α is an action and $S \subseteq \Delta_0$, then $\mathsf{Done}_S(\alpha)$ is a formula.

The intuitive reading of $\mathsf{Done}_S(\alpha)$ is: *if we restrict the actions of the component to those appearing in S, then the last action executed was* α. Note that the classic $\mathsf{Done}()$ operator is just $\mathsf{Done}_{\Delta_0}(\alpha)$; for the sake of simplicity, sometimes we write $\mathsf{Done}(\alpha)$ instead of $\mathsf{Done}_{\Delta_0}(\alpha)$. That is, we have similar formulae to [5], but we add indexed permissions and a more general version of the $\mathsf{Done}()$.

We also add a logical constant to predicate about the initial state of a system:

- B is a formula. symbols]B

Note that in the logic defined in [5] the predicate B is not needed since it is expressed by $\neg\mathsf{Done}(\mathbf{U})$; however, here, since we consider transitions which can be labeled with external events, this formula does not denote the initial instant of the system, but the initial state of the current component.

We also introduce some changes to the semantic structures to give the semantics of this variation of the logic. Basically, we follow the ideas of [11], 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 [1]). Intuitively, each action produces a (finite) set of events during the execution of the system (the events that this action "observes"), and also there are other events produced by actions from other components or from the environment.

Definition 1 (models)**.** *Given a language $L = \langle \Phi_0, \Delta_0, V_0, I_0\rangle$, an L-Structure is a tuple: $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \{\mathcal{P}^i \mid i \in I_0\}\rangle$ where:*

- *$\mathcal{W}$, is a set of worlds.*

- *$\mathcal{R}$, is an $\mathcal{E}$-labeled 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 a function:*
  - *For every $p \in \Phi_0 : \mathcal{I}(p) \subseteq \mathcal{W}$*
  - *For every $\alpha \in \Delta_0 : \mathcal{I}(\alpha) \subseteq \mathcal{E}$, and $\mathcal{I}(\alpha)$ is finite.*

  In addition, the interpretation $\mathcal{I}$ has to satisfy the following properties:

  I.1 *For every $\alpha_i \in \Delta_0$: $|\mathcal{I}(\alpha_i) - \bigcup\{\mathcal{I}(\alpha_j) \mid \alpha_j \in (\Delta_0 - \{\alpha_i\})\}| \leq 1$.*

  I.2 *For every $e \in \mathcal{I}(a_1 \sqcup \cdots \sqcup a_n)$: if $e \in \mathcal{I}(\alpha_i) \cap \mathcal{I}(\alpha_j)$, where $\alpha_i \neq \alpha_j \in \Delta_0$, then:*
  $\cap\{\mathcal{I}(\alpha_k) \mid \alpha_k \in \Delta_0 \wedge e \in \mathcal{I}(\alpha_k)\} = \{e\}$.
- *each $\mathcal{P}^i \subseteq \mathcal{W} \times \mathcal{E}$ is a relation which indicates which event is permitted in which world with respect to permissions with index i.*

$\square$

Roughly speaking, the structure gives us a labeled transition system, whose labels are events, which are produced by some action or they could also be external events. Note that we have a set of events, but actions are only interpreted over finite subsets, whose intersections satisfy the Hausdorff condition, i.e., we require that every one-point set can be generated from the actions of the component. 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 [5] is given only in terms of standard models.

We use maximal traces to give the semantics of the temporal operators, and in the following we use the notation introduced in [6], Since, in a trace, we have events that do not belong to the actual components, we need to distinguish between those events generated by the component being specified and those which are from the environment. Given language L, a L-structure M and a maximal path π in M, we define the set:

$$\mathrm{Loc}_L(\pi) = \{i \mid \pi(i) \in \mathcal{I}(a_1 \sqcup \cdots \sqcup a_n)\} \cup \{0\}$$

symbols]Loc_L (where $a_1, \ldots, a_n$ are all the primitive actions of L), i.e., this set contains all the positions of π where events occurs that are observed by some action in L. Obviously, this set is totally ordered by the usual relationship $\leq$. Also we consider a restricted version of this set; given a set $\{a_1, \ldots, a_m\} \subseteq \Delta_0$, we define:

$$Loc_{\{a_1,\ldots,a_m\}}(\pi) = \{i \mid \pi(i) \in \mathcal{I}(a_1 \sqcup \cdots \sqcup a_m)\}$$

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 $\max_p(S)$. Using these concepts, we define the semantics in a similar way to that used in [5], but taking into account the separation between local and external events.

Definition 2. *Given a trace* $\pi = s_0 \overset{e_0}{\to} s_1 \overset{e_1}{\to} s_2 \overset{e_2}{\to} ... \in \Sigma^*(w)$, *we define the relation* $\vDash_L$ *as follows:*

- *If* $p_j \in \Phi_0 \cup V_0$, *then* $\pi, i, M \vDash_L p_j \overset{\mathsf{def}}{\iff} p_j \in \mathcal{I}(\pi_i)$.
- $\pi, i, M \vDash_L \mathsf{P}^i(\alpha) \overset{\mathsf{def}}{\iff} \forall e \in \mathcal{I}(\alpha) : \mathcal{P}^i(w, e)$.
- $\pi, i, M \vDash_L \mathsf{P}^i_{\mathsf{w}}(\alpha) \overset{\mathsf{def}}{\iff} \exists e \in \mathcal{I}(\alpha) : \mathcal{P}^i(w, e)$.
- $\pi, i, M \vDash_L \neg\varphi \overset{\mathsf{def}}{\iff}$ *not* $\pi, i, M \vDash_L \varphi$.
- $\pi, i, M \vDash_L \varphi_1 \to \varphi_2 \overset{\mathsf{def}}{\iff}$ *either not* $\pi, i, M \vDash_L \varphi_1$ *or* $\pi, i, M \vDash_L \varphi_2$.
- $\pi, i, M \vDash_L \mathsf{Done}_S(\alpha) \overset{\mathsf{def}}{\iff} \exists j : j = max_{<i}(Loc_S(\pi)) \wedge e_j \in \mathcal{I}(\alpha)$.
- $\pi, i, M \vDash_L [\alpha]\varphi \overset{\mathsf{def}}{\iff} \forall \pi' = s'_0 \overset{e'_0}{\to} s'_1 \overset{e'_1}{\to} ... \in \Sigma^*(w)$ *such that* $\pi[0, i] \prec \pi'$, *if* $j = min_{>i}(Loc(\pi'))$, *and if* $e'_j \in \mathcal{I}(\alpha)$, *then* $\pi', j, M \vDash_L \varphi$.
- $\pi, i, M \vDash_L \mathsf{AN}\varphi \overset{\mathsf{def}}{\iff}$ *if* $i = \#\pi$, *then* $\pi, i, M \vDash \varphi$. *If* $i \neq \#\pi$, *then* $\forall \pi' \in \Sigma^*(w) : \pi[0..i] \prec \pi'$:, *if* $j = min_{>i}(Loc(\pi'))$, *then* $\pi', j, M \vDash \varphi$.
- $\pi, i, M \vDash_L \mathsf{A}(\varphi_1 \, \mathcal{U} \, \varphi_2) \overset{\mathsf{def}}{\iff}$ *if* $i = \#\pi$, *then* $\pi, i, M \vDash \varphi_2$. *If* $i \neq \#\pi$, *then* $\forall \pi' \in \Sigma^*(w) : \pi[0..i] \prec \pi'$ *we have that* $\exists j \in Loc((\pi')^i) : \pi', j, M \vDash \varphi_2$ *and* $\forall i \leq k \leq j : k \in Loc((\pi')^i)$, *then* $\pi', k, M \vDash \varphi_1$.
- $\pi, i, M \vDash_L \mathsf{E}(\varphi_1 \, \mathcal{U} \, \varphi_2) \overset{\mathsf{def}}{\iff}$ *if* $i = \#\pi$, *then* $\pi, i, M \vDash \varphi_2$. *If* $i \neq \#\pi$, *then* $\exists \pi' \in \Sigma^*(w) : \pi[0..i] \prec \pi'$ *such that* $\exists j \in Loc((\pi')^i) : \pi', j, M \vDash \varphi_2$ *and* $\forall i \leq k \leq j : k \in Loc((\pi')^i)$, *then* $\pi', k, M \vDash \varphi_1$.

□

We say that $\vDash_L \varphi$, if $\pi, i, M \vDash \varphi$, for every model M and path π. (In this report, when we use the symbol $\vDash$, we refer to this relationship and not the one defined in [5])

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 done in [11] and [12]) that external events do not produce changes in local variables. In [12] 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 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 [12] and we define *locus models* which have the property of generating locus traces. We need to investigate the model theory of our logic more deeply to be able to define this concept.

We have presented an axiomatic system for an ealier version of this temporal 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. $\langle\alpha\rangle\bot \leftrightarrow \bot$

   A2. $[\emptyset]\varphi$

   A3. $\langle\alpha\rangle\varphi \wedge [\alpha]\psi \rightarrow \langle\alpha\rangle(\varphi \wedge \psi)$

   A4. $[\alpha \sqcup \alpha']\varphi \leftrightarrow [\alpha]\varphi \wedge [\alpha']\varphi$

   A5. $[\alpha]\varphi \rightarrow [\alpha \sqcap \alpha']\varphi$

   A6. $\mathsf{P}(\emptyset)$

   A7. $\mathsf{P}(\alpha \sqcup \beta) \leftrightarrow \mathsf{P}(\alpha) \wedge \mathsf{P}(\beta)$

   A8. $\mathsf{P}(\alpha) \vee \mathsf{P}(\beta) \rightarrow \mathsf{P}(\alpha \sqcap \beta)$

   A9. $\neg\mathsf{P_w}(\emptyset)$

   A10. $\mathsf{P_w}(\alpha \sqcup \beta) \leftrightarrow \mathsf{P_w}(\alpha) \vee \mathsf{P_w}(\beta)$

   A11. $\mathsf{P_w}(\alpha \sqcap \beta) \rightarrow \mathsf{P_w}(\alpha) \wedge \mathsf{P_w}(\beta)$

   A12. $\mathsf{P}(\alpha) \wedge \alpha \neq \emptyset \rightarrow \mathsf{P_w}(\alpha)$

   A13. $\mathsf{P_w}(\gamma) \rightarrow \mathsf{P}(\gamma)$, where $[\gamma]$ is an atom in Δ_0/Φ_{BA}.

   A14. $\mathsf{O}(\alpha) \leftrightarrow \mathsf{P}(\alpha) \wedge \neg\mathsf{P_w}(\overline{\alpha})$

   A15. $[\alpha]\varphi \leftrightarrow \neg\langle\alpha\rangle\neg\varphi$

   A16. $(a_1 \sqcup ... \sqcup a_n) =_{act} \mathbf{U}$

   A17a. $\alpha =_{act} \alpha' \rightarrow [\beta](\alpha =_{act} \alpha')$

   A17b. $\langle\beta\rangle(\alpha =_{act} \alpha') \rightarrow \alpha =_{act} \alpha'$

   A18. $\langle\gamma\rangle\varphi \rightarrow [\gamma]\varphi$, where $[\gamma]$ is an atom of Δ_0/Φ_{BA}

   BA. $\varphi[\alpha] \wedge (\alpha =_{act} \alpha') \rightarrow \varphi[\alpha/\alpha']$

For the temporal extension of the logic consider the axioms above plus:

TempAx1. $\langle \mathbf{U} \rangle \top \rightarrow (\mathsf{AN}\varphi \leftrightarrow [\mathbf{U}]\varphi)$

TempAx2. $[\mathbf{U}]\bot \rightarrow (\mathsf{AN}\varphi \leftrightarrow \varphi)$

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. $\mathsf{B} \rightarrow \neg\mathsf{Done}_S(\alpha)$

TempAx10. $[\mathbf{U}]\neg\mathsf{B}$

TempAx11. $\mathsf{Done}_S(\alpha) \rightarrow [\overline{\bigsqcup_{a\in S} a}]\mathsf{Done}_S(\alpha)$

TempAx12. $\neg\mathsf{Done}_S(\alpha) \rightarrow [\overline{\bigsqcup_{a\in S} a}]\neg\mathsf{Done}_S(\alpha)$

And the following deduction rules:

- Rules given in [5] for the propositional part of the logic.

TempRule1. if $\vdash \mathsf{B} \rightarrow \varphi$ and $\vdash \varphi \rightarrow [\mathbf{U}]\varphi$, then $\vdash \varphi$

TempRule2. if $\vdash \varphi$, then $\vdash \mathsf{AG}\varphi$

TempRule3. if $\vdash \varphi \rightarrow (\neg\psi \wedge \mathsf{EN}\varphi)\}$, then $\vdash \varphi \rightarrow \neg\mathsf{A}(\varphi'\,\mathcal{U}\,\psi)$

TempRule4. if $\vdash \varphi \rightarrow (\neg\psi \wedge \mathsf{AN}(\varphi \vee \neg\mathsf{E}(\varphi'\,\mathcal{U}\,\psi)))$, then $\vdash \varphi \rightarrow \neg\mathsf{E}(\vartheta\,\mathcal{U}\,\psi)$

TempRule5. if $\vdash \neg\mathsf{Done}(\mathbf{U}) \rightarrow \mathsf{AG}\varphi$, then $\vdash \varphi$

The new axioms are **TempAx6-TempAx12**. 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 relativised $\mathsf{Done}()$ 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 $\mathsf{Done}()$ operator as defined in [7].

2.1 A Touch of Model Theory.

For the next definition we fix two structures $M_1 = \langle \mathcal{W}_1, \mathcal{R}_1, \mathcal{E}_1, \mathcal{I}_1, \mathcal{P}_1, w_1 \rangle$, and $M_2 = \langle \mathcal{W}_2, \mathcal{R}_2, \mathcal{E}_2, \mathcal{I}_2, \mathcal{P}_2, w_2 \rangle$, then we define the notion of morphism between M_1 and M_2.

Definition 3. *A morphism $m : M_1 \to M_2$ between M_1 and M_2 is a pair of functions $\langle f_W : \mathcal{W}_1 \to \mathcal{W}_2, f_E : \mathcal{E}_1 \to \mathcal{E}_2 \rangle$, which satisfies:*

M0 $f_W(w_1) = w_2$.

M1 *For every $p_i \in \Phi_0$ and $w \in \mathcal{W}_1$, if $p_i \in \mathcal{I}_1(w)$, then $p_i \in \mathcal{I}_2(f_W(w))$.*

M2 *For every $e \in \mathcal{E}_1$ and $a_i \in \Delta_0$, if $e \in \mathcal{I}_1(a_i)$, then $f_E(e) \in \mathcal{I}_2(a_i)$.*

M3 *For every $e \in \mathcal{E}_1$ and $w, w' \in \mathcal{W}_1$, if $w \overset{e}{\to} w' \in \mathcal{R}_1$, then $f_W(w) \overset{f_E(w)}{\to} f_W(w') \in \mathcal{R}_2$.*

M4 *For every $e \in \mathcal{E}_1$ and $w \in \mathcal{W}_1$, if $\langle w, e \rangle \in \mathcal{P}_1^i$, then $\langle f_W(w), f_E(e) \rangle \in \mathcal{P}_2^i$.*

□

We say that a morphism $m = \langle f_W, f_E \rangle$ is surjective, if f_W and f_E are onto, and we say that m is injective if both f_W and f_W are injective. To define the notion of isomorphism we need to introduce the concept of *strong morphism* (where we follow the terminology used in the model theory of modal logics [2]).

Definition 4. *A morphism $m : M_1 \to M_2$ is strong iff the conditions **M1-M4** are equivalences.* □

Then, a morphism $m = \langle f_W, f_E \rangle : M_1 \to M_2$ is an isomorphism if m is a strong morphism and the components f_W and f_E are bijections, we denote this situation by $M_1 \cong M_2$. Note that, given a morphism $m : M_1 \to M_2$ and given a trace

$$\pi = w \overset{e_1}{\to} w_1 \overset{e_2}{\to} \dots$$

we can define a corresponding trace:

$$m(\pi) = f_W(w) \overset{f_E(e_1)}{\to} f_W(e_2) \dots$$

If $m = \langle f_W, f_E \rangle : M_1 \to M_2$ is an isomorphism, then $m^{-1} = \langle f_W^{-1}, f_E^{-1} \rangle : M_2 \to M_1$ is also an isomorphism, and it is the inverse of m, i.e., $m \circ m^{-1} = id_{M_1}$ and $m^{-1} \circ m = id_{M_2}$.

It is straightforward to prove that isomorphic models are elementarily equivalent, that is:

Theorem 1. *Given two models M_1 and M_2, if $M_1 \cong M_2$, then $M_1 \vDash \varphi$ iff $M_2 \vDash \varphi$, for every formula φ of L.*

However, isomorphism is a strong condition, and we are interested in situations when the models are not exactly isomorphic but where the structure of one of them is somehow preserved by the other. The notion of bisimulation (introduced in the context of process algebra [19]) has been shown to be useful to show equivalence of modal formulae with respect to Kripke semantics [21], and with respect to temporal logics [3] and [9]. Here, we describe a notion of bisimulation that is useful for our purposes and is related to branching bisimulation [22] and stuttering bisimulation [16]; note that in the latter case, the notion of bisimulation is defined over non-labeled systems, while our notion of bisimulation is defined over two different labeled transition systems. Using the terminology of [9], we can say that the bisimulation presented later on is a *sensitive divergence* bisimulation, since it distinguishes between processes which diverge by non-local events; we come back to this topic later on.

Recall that, in a given 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 introduce some notation useful for the following sections. We say $w \overset{\epsilon}{\Rightarrow} w'$, if there exists a path $w \overset{e_0}{\rightarrow} w_1 \overset{e_1}{\rightarrow} w_2 \overset{e_2}{\rightarrow} \dots \overset{e_n}{\rightarrow} w_n$, such that e_i is non-local for every $0 \leq i \leq n$. We say that $w \overset{\infty}{\Rightarrow}$ when there is an infinite path from w: $w \overset{e_0}{\rightarrow} w_1 \overset{e_1}{\rightarrow} \dots$, such that every e_i is non-local. Furthermore, we say $w \overset{e}{\Rightarrow} w'$ (where e is local) if $w \overset{\epsilon}{\Rightarrow} w''$ and $w'' \overset{e}{\rightarrow} w'$.

Given two structures M and M', such that $\mathcal{E} = \mathcal{E}'$ (i.e., they have the same events), assume $\mathcal{I}(\alpha) = \mathcal{I}'(\alpha)$ for any α (the interpretation of every action gives us the same events). We say that a relationship $Z \subseteq W \times W'$ is a *local bisimulation* iff:

- If wZv, then $L(w) = L(v)$.
- If wZv, and $w \overset{\infty}{\Rightarrow}$, then either $v \overset{\infty}{\Rightarrow}$ or there is a v' such that $v \overset{\epsilon}{\Rightarrow} v'$ and v' has no successors by $\rightarrow$.
- if wZv and $w \overset{e}{\rightarrow} w'$. then $w'Zv$ if e is non-local. Otherwise we have some v' such that $v \overset{e}{\rightarrow} v'$ 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. (Note that the composition of two local bisimulations is a local bisimulation, and the identity relation is a local bisimulation.) In branching bisimulation (as defined in [9]), 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 *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 [9], it is shown that CTL^*-X (CTL^* without the next operator) cannot distinguish between Kripke structures which are DBSB (divergent blind 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. On the other hand, 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).

Using bisimilarity, we define the notion of a *locus* model (following the terminology of [12] where locus models are introduced in a linear temporal logic).

Definition 5. *We say that a structure M is a locus iff there is a local bisimulation between M and a standard model M'.* □

It is worth noting that in this work we are not interested in comparing the expressivity of our logic with respect to the notion of bisimilarity introduced above (as done in [9]). Instead, we use this notion of bisimulation to formalize the notion 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 behavior 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.

We extend the definition of bisimulation to paths.

Definition 6. *Given a path $\pi = w_0 \xrightarrow{e_0} w_1 \xrightarrow{e_1} \ldots$ in M and a path $\pi' = v_0 \xrightarrow{d_0} v_1 \xrightarrow{d_1} \ldots$ in M', and a local bisimulation between M and M' such that $M \sim_Z M'$, we say that $\pi Z \pi'$, iff when $w_i Z v_i$, then:*

- *if we have $w_i \xrightarrow{e_1} \ldots \xrightarrow{e_n} w_n$ in π, with e_j non-local for $0 \leq j \leq n$, then we have $v_j \xrightarrow{d_1} \ldots \xrightarrow{d_m} v_m$ in π', with d_l non-local for every $0 \leq l \leq m$, 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': $v_j \xrightarrow{d_1} \ldots \xrightarrow{e} v_m$ such that for all $1 \leq l < m$, d_l are non-local, and $w_n Z v_m$.*

- *we also have $\pi' Z^{\smile} \pi$.*

$\square$

This is similar to the definition of stuttering equivalence, but taking into account the labels. Our first property about paths and local bisimulation is the following.

Theorem 2. *If $\pi[0..i] Z \pi'[0..j]$, then there exists a $\pi'[0..j] \preceq \pi_2$ such that $\pi Z \pi_2$.*
Proof. *If from position i in π we have an infinite sequence of non-local events, then $\pi_i \overset{\infty}{\Rightarrow}$ and therefore, by definition of local bisimulation, $\pi'_j \overset{\infty}{\Rightarrow}$. Thus we have some fullpath π_2 such that $\pi Z \pi_2$. Otherwise, we have some e and k such that $\pi_i \overset{e}{\Rightarrow} \pi_k$ in π; but, since $\pi_i Z \pi'_j$ we can find a state $v_{k'}$ in M' such that $\pi'_j \overset{e}{\Rightarrow} v_{k'}$. 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 $\pi Z \pi_2$.* $\blacksquare$

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:

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

where $\pi = w_0 \xrightarrow{e_0} w_1 \xrightarrow{e_1} \dots$. In the same way we can define a function $f_{\pi'}$. An useful property of these functions is the following.

Property 1. *If $\pi Z \pi'$, then $\pi_i Z \pi'_{f_\pi(i)}$, for every position i of π.*
Proof. *The proof is by induction; for the basis is straightforward. For the inductive case, suppose that $\pi_i Z \pi'_{f_\pi(i)}$; if $\pi_i \xrightarrow{e_i} \pi_{i+1}$ 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, $f_\pi(i+1) = \mathsf{min}_{>f_\pi(i)}(Loc(\pi'))$, and by definition of bisimulation between paths we get $\pi_{i+1} Z \pi_{f_\pi(i+1)}$.*
$\blacksquare$

Property 2. *If $\pi Z \pi'$, $\#Loc(\pi[0..i]) = \#Loc(\pi'[0..f_\pi(i)])$.*
Proof. *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 $\pi_i \xrightarrow{e_i} \pi_{i+1}$ 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 $\#Loc(\pi[0..i+1]) = \#(Loc(\pi[0..i]) \cup \{e_i\})$ and then we have $\pi'_{f_\pi(i)} \overset{e_i}{\Rightarrow} \pi'_{f_\pi(i+1)}$, and then $\#Loc(\pi'[0..f_\pi(i+1)]) = \#(Loc(\pi'[0..f_\pi(i)]) \cup \{e_i\}) = \#Loc(\pi[0..i+1])$. $\blacksquare$

A useful corollary of the above property is the following.

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

■

By definition of f_π and $f_{\pi'}$, we obtain the following properties which resemble the properties of Galois connections.

Property 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 fullpaths satisfy the same properties:

Theorem 3. *If $\pi Z \pi'$, then, for all position i, $\pi, i, M \vDash \varphi \Leftrightarrow \pi', f_\pi(i), M \vDash \varphi$.*
Proof. *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 $k = \mathsf{max}_{<i}(Loc_S(\pi))$, we have that $e_k \in \mathcal{I}(\alpha)$, and $\pi_{k-1} Z \pi'_{f_\pi(k-1)}$. But then we have $\pi'_{f_\pi(k-1)} \overset{e}{\Rightarrow} \pi'_{f_\pi(k)}$, and $\pi'_{f_\pi(k)} \overset{\epsilon}{\Rightarrow} \pi'_{f_\pi(i)}$, 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 $k = \mathsf{min}_{>i}(Loc(\pi_2))$ such that $(\pi_2)_k \in \mathcal{I}(\alpha)$ and $\pi_2, k, M' \nvDash \varphi$. By theorem 2, we know that we have a fullpath $\pi_1 \succeq \pi[0..i]$ such that $\pi_1 Z \pi_2$. By the definition of bisimulation between paths we know that if $(\pi_2)_{f_{\pi_1}(i)} \overset{e}{\Rightarrow} (\pi_2)_k$ in π_2, then $(\pi_1)_i \overset{e}{\Rightarrow} (\pi_1)_{f_{\pi_2}(k)}$ in π_1. Applying induction on the symmetric statement of the theorem, we get $\pi_2, f_{\pi_2}(k), M \nvDash \varphi$ which is a contradiction. The other direction is similar.*

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

If $\pi, i, M \vDash \mathsf{A}(\varphi\ \mathcal{U}\ \psi)$, suppose $\pi', f_\pi(i), M' \nvDash \mathsf{A}(\varphi\ \mathcal{U}\ \psi)$. Then, if $\pi', f_\pi(i), M' \nvDash \varphi$, we get a contradiction. Otherwise, we must have a fullpath $\pi_2 \succeq \pi'[0..f_\pi(i)]$, such that for every $j \in Loc(\pi_2)$ we have $\pi_2, j, M' \nvDash \psi$. 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 $\pi_1, k, M \vDash \psi$, for this k we have that $\pi_2, f_\pi(k), M' \vDash \psi$, by induction. But note that $\pi_2(f_\pi(k)) \in Loc(\pi_2)$ which gives us a contradiction, and therefore $\pi', f_\pi(i), M' \vDash \mathsf{A}(\varphi\ \mathcal{U}\ \psi)$. The other direction is similar.

If $\pi, i, M \vDash \mathsf{E}(\varphi\ \mathcal{U}\ \psi)$, and suppose $\pi', f_\pi(i), M' \nvDash \mathsf{E}(\varphi\ \mathcal{U}\ \psi)$, then if $\pi', f_\pi(i), M' \nvDash \varphi$ and $\pi', f_\pi(i), M', \nvDash \psi$, we get a contradiction. Otherwise, *we have that for every path $\pi'' \succeq \pi'[0..f_\pi(i)]$ and for every $k \in Loc(\pi_2^{f_\pi(i)})$, $\pi'', k, M' \nvDash \psi$ holds . Note that we have a π_2 in M' such that $\pi_1 Z \pi_2$ (where π_1 is that fullpath mentioned above), and then by induction $\pi_2, f_\pi(i), M' \vDash \psi$. Furthermore, note that $\pi_2(f_\pi(i))$ is a local event, which contradicts the assumption above, and therefore $\pi', f_\pi(i), M' \vDash \mathsf{E}(\varphi \, \mathcal{U} \, \psi)$. The other direction is similar.* ■

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

Theorem 4. *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 fullpath 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.* ■

2.2 Locus Models.

At the beginning of section 2, we introduced non-standard models (i.e., those models which have "external" events). However, not all non-standard models are useful; we want that the external events preserve local variables, that is, the events not generated by any of the actions in the component have to be silent, in some sense. In [12], with the same purpose in mind, the notion of locus trace is introduced. A *locus* trace is one in which, after executing a non-local event, the local variables retain their value. However, since we have a branching time logic and a modal logic, in our logic it is not enough to just put restrictions on traces. We need to take into account the branching occurring in the semantic structures, i.e., we need a more general notion of locus model.

Roughly speaking, locus models are those which are local 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.

Definition 7. *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 5. *If M is a locus structure, then $M \vDash \varphi$ iff there is some standard structure M' such that $M' \vDash \varphi$.*
Proof. *If M is standard the result follows. Otherwise we use theorem 4.* ■

Using this theorem we can prove that the axiomatic system described in section 2 is sound with respect to locus structures.

Theorem 6. *The axiomatic system of section 2 is sound with respect to locus structures and the relation* $\vDash$ *defined in section 2.*
Proof. *Note that, if we only take into account standard structures, the definition of* $\vDash$ *coincides with the definition given in [7]. Therefore, axioms* **A1** – **A18** *are sound with respect to standard models and then, by theorem 5, 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 straighforward using the definition of* B *and the relativized* $\mathsf{Done}()$. ■

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 is a translation between two languages.

Definition 8. *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:*

- *An injective mapping* $f : \Delta_0 \to \Delta'_0$ *between the actions of component* C *and the actions of* C'.
- *A mapping* $g : \Phi_0 \to \Phi'_0$ *between the propositions of* L *and the predicates of* L'.
- *An injective mapping* $h : V_0 \to V'_0$, *between the violations of* L *and the violations of* L'.
- *An injective mapping* $i : I_0 \to I'_0$.

□

For the sake of simplicity, we denote the application of any of these functions using the name of the mapping, e.g., instead of writing $f(a_i)$ we write $\tau(a_i)$.

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 (which straightforwardly satisfy associativity). Now, given a translation, we can extend this translation to formulae (actually we can describe a functor grammar which reflects these facts, as done in Institutions [14] or π-Institutions [13]). 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. First, we need to define how the translation behaves with respect to action terms:

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

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. Now, the extension to formulae is as follows:

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

Note, τ is a function which preserves logical symbols. In other words, using translations between signatures, we can define morphisms between formulae, and therefore we can define interpretations between theories (in the sense of [10]); we deal with this issue in the next section.

Note that, given a translation $\tau : L \to L'$ and given a L'-structure M, it is straightforward to define the restriction of $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \{\mathcal{P}^i \mid i \in I_0\}\rangle$ with respect to τ (or its reduct as it is called in model theory [8]), as follows:

Definition 9. *Given a translation $\tau : L \to L'$ and a L'-structure M we can define a L-structure $M|_\tau$ as follows:*

- $\mathcal{W}|_\tau = \mathcal{W}$.
- $\mathcal{E}|_\tau = \mathcal{E} - \{e \in \mathcal{I}(\tau(\mathbf{U})) \cap \mathcal{I}(\Delta'_0 - \tau(\Delta_0))\}$.
- $\mathcal{I}|_\tau(a_i) = \{e \in \mathcal{I}(a) \mid e \in \mathcal{E}|_\tau\}$*, for every* $a_i \in \Delta'_0$.

- $\mathcal{I}|_\tau(p_i) = I(\tau(p_i))$, *for every* $p_i \in \Phi'_0$.
- $\mathcal{R}|_\tau = \{w \xrightarrow{e} w' \in \mathcal{R} \mid e \in \mathcal{E}\}$.
- $\mathcal{P}^i|_\tau(w, e) \Leftrightarrow P^{\tau(i)}(w, e)$.
- $w_0|_\tau = w_0$.

$\square$

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, 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 [14] to define logical systems; we investigate the institutional aspects of our logic later on. An important problem with restrictions is 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 with 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 10. *Given a translation $\tau : L \to L'$ and a L'-structure M, we say that M is a τ-locus iff:*

- $M \vDash \mathsf{ind}(\tau)$.
- $M \vDash \mathsf{atom}(\tau)$
- *The restriction $M|_\tau$ 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. We have obtained a semantical characterization of structures which respect the local behavior 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 example, in modal logics the class of Kripke structures which are reflexive are characterized by adding the axiom T to K (called a T system [15]); in a similar way the transitive Kripke structures are characterized by adding the axiom 4 (called a $K4$ system). So, a natural question is: is there some way of characterizing locus models? We shall prove that we have an affirmative answer to this question. 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. *Given a τ-locus model M, for every fullpath π' of M such that $\pi' \succeq \pi[0..i]$, there is fullpath $\pi'' \succeq \pi[0..i]$ in M such that π'' also is a fullpath of $M|_\tau$ and: $\pi', i, M \vDash \tau(\varphi)$ iff $\pi'', i, M \vDash \tau(\varphi)$.*
proof *The proof is by induction on φ and using the properties of independence and atomicity.* ■

Property 5. *If $\tau : L \to L'$, and M is a L' structure which is a τ-locus, then if $\pi, i, M|_\tau \vDash \varphi$, and $\pi(i)$ is non-local, then $\pi, i+1, M|_\tau \vDash \varphi$.*
Proof. *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.* ■

Another useful property is the following:

Property 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 $M|_\tau$.*
Proof. *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.* ■

First, let us prove that local properties are preserved by τ-locus structures.

Theorem 7. *Let $\tau : L \to L'$ be translation and M a L'-structure. If M is a τ-locus, then for fullpath π of $M|_\tau$, $\pi, i, M \vDash \tau(\varphi)$ iff $\pi, i, M|_\tau \vDash \varphi$, for any formulae φ of L.*

Proof. *The proof is by induction on φ.*

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

Ind. Case. If $\pi, i, M \vDash \tau([\alpha]\varphi)$ which is equivalent to $\pi, i, M \vDash [\tau(\alpha)]\tau(\varphi)$, and now suppose that $\pi, i, M|_\tau \nvDash [\alpha]\varphi$. From here we have that there exists a path $\pi' \succeq \pi[0..i]$ such that $\pi, i+1, M|_\tau \nvDash \varphi$, where $\pi'(i+1) \in \mathcal{I}|_\tau(\alpha)$. Now we have the same trace in M, which gives us a contradiction by induction. If $\pi, i, M|_\tau \vDash [\alpha]\varphi$, suppose $\pi, i, M \nvDash [\tau(\alpha)]\tau(\varphi)$, and then we have a $\pi' \succeq \pi[0..i]$ (noting that, if π' is not a fullpath of $M|_\tau$ then, applying property 4, we can find an equivalent path which belongs to this model) such that $\pi, i, M \nvDash \tau(\varphi)$ and $\pi'(i) \in \mathcal{I}(\tau(\alpha))$. By definition of reduction, we have that $\pi'(i) \in \mathcal{I}|_\tau(\alpha)$, which applying induction, gives us a contradiction, and therefore $\pi, i, M \vDash [\tau(\alpha)]\tau(\varphi)$.

Suppose $\pi, i, M \vDash \tau(\mathsf{AN}(\varphi))$ and $\pi, i, M|_\tau \nvDash \mathsf{AN}\varphi$. Then, if i is the last position of π, then we have $\pi, i, M|_\tau \nvDash \varphi$, which gives us a contradiction, since by induction this implies $\pi, i, M \nvDash \varphi$. If i is not the last position of π, then, for $k = \mathsf{min}_{>i}(Loc_L(\pi))$, we have $\pi, k, M|_\tau \nvDash \varphi$, note that $\pi_i \overset{e}{\Rightarrow} \pi_k$ 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 $\pi, k, M \nvDash \mathsf{Done}(a_1 \sqcup \cdots \sqcup a_n) \to \varphi$, which contradicts what we said above, and therefore $\pi, i, M|_\tau \vDash \mathsf{AN}\varphi$. The other direction is similar.

Suppose that $\pi, i, M \vDash \tau(\mathsf{A}(\varphi\ \mathcal{U}\ \psi))$ and $\pi, i, M|_\tau \nvDash \mathsf{A}(\varphi\ \mathcal{U}\ \psi)$. If $\pi, i, M|_\tau \nvDash \varphi$ and $\pi, i, M|_\tau \nvDash \psi$ (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 $\pi', k, M|_\tau \nvDash \psi$, for every $k \in Loc_L(\pi^i)$, then note that for this π' in M we have $\pi, i, M \nvDash \tau(\psi)$ (by induction) and from here if a position $j \leq i$ is reached by a non-local event for L we have, by property 5 and induction, that $\pi', j, M \vDash \tau(\psi)$, and if it is local, then we have by the supposition above that $\pi, j, M \vDash \tau(\psi)$, i.e., for every $j \geq i$ $\pi, j, M \nvDash \tau(\psi)$, which contradicts our initial assumption, and therefore $\pi, i, M|_\tau \vDash \mathsf{A}(\varphi\ \mathcal{U}\ \psi)$. The other direction is similar.

If $\pi, i, M \vDash \tau(\mathsf{E}(\varphi\ \mathcal{U}\ \psi))$, then we have some $\pi' \succeq \pi[0..i]$ such that there is a k such that $\pi, k, M \vDash \tau(\psi)$ where $k \geq i$, and for every $j \in Loc_{L'}(\pi')$ such that $i \leq j \leq k$, we have $\pi', j, M \vDash \tau(\varphi)$; then, since $Loc_L(\pi') \subseteq Loc_{L'}(\pi')$ and using induction, we have that for every $j \leq k$ such that $j \in Loc_L(\pi')$, $\pi, j, M|_\tau \vDash \varphi$, and $\pi, k, M|_\tau \vDash \psi$. Now if $k \notin Loc_L(\pi')$, then, by proposition 4, it must be a $k' \in Loc(\pi')$ such that $k' \leq k$ and $\pi', k', M|_\tau \vDash \psi$. On the other hand, if $\pi, i, M|_\tau \vDash \mathsf{E}(\varphi \in \psi)$, then we have some $\pi' \succeq \pi[0..i]$ such *that $\pi', k, M|_\tau \vDash \psi$, where $k \in Loc_L(\pi')$, and for every $i \leq j \leq k$ with $j \in Loc_L(\pi')$ we have $\pi', j, M|_\tau \vDash \varphi$. Note that, using property 4, we have that for every position $j \in Loc_{L'}(\pi')$ such that $i \leq j \leq k$, we have that $\pi', j, M \vDash \tau(\varphi)$ (since, if it is a local event for L, we show above that it satisfies φ, otherwise it preserves the property), and $k \in Loc_{L'}(\pi')$ and then $\pi, k, M \vDash \psi$ by induction.* ■

We have a semantic characterization of τ-locus models, but since we want to use deductive systems, we need a syntactic characterization of this class of models. 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. Note that, in [12], a similar set of axioms is proposed, although in that case it is a finite set, since that work uses a linear temporal logic, and therefore preserving only the propositions is enough for having a good notion of locality. However 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 behavior 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 τ-loci L'-structures.

Theorem 8. *Given a translation $\tau : L \to L'$, then a L'-structure is a τ-locus iff $M \vDash Loc(\tau)$.*
Proof. *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 and theorem 7 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 collection of states:

$$[\epsilon] = \{v \mid w_0 \overset{\epsilon}{\Rightarrow} v\} \cup \{w_0\}$$

and:

$$[es \centerdot e] = \{v \mid \exists z, v' : (z \in [es]) \wedge (z \overset{e}{\Rightarrow} v') \wedge ((v' \overset{\epsilon}{\Rightarrow} v) \vee v = v')\}.$$

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

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

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 $w \overset{\infty}{\Rightarrow}$, then we know that $[es]$ cannot diverge by non-local events. The only possibility is that there is no e such that $[es] \xrightarrow{e} [es \centerdot e]$; 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(τ), 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 that $L(w) = L(w') = L([es])$. If $w \xrightarrow{e} w'$ and e is local, then we know that $w' \in [es \centerdot e]$ and then $w'Z[es \centerdot e]$, and furthermore $[es] \xrightarrow{e} [es \centerdot e]$, by definition. Now, if $[es]Z^{\smile}w$, it is worth noting that $M^{\#}$ does not have any divergence via non-local events. If $[es] \xrightarrow{e} [es \centerdot e]$, we have some $w' \in [es \centerdot e]$ (by definition), and note that we have some $v \in [es]$

and $w'' \in [es \centerdot e]$ such that $v \overset{e}{\Rightarrow} w''$ and $w'' \overset{\epsilon}{\Rightarrow} w'$. 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 $v \overset{e}{\Rightarrow} w''$ 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 $v' \in [es \centerdot e]$ such that $w \xrightarrow{e} v''$, which finishes the proof. ■

It is worth remarking again that 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 over them is *modus ponens*. Instead, $\vdash^L_S \varphi$ says that, if we extend our axiomatic system with the formulae of S, then we can prove φ. This can be thought of as a *global* deduction (note that in this case we can apply any rule to the members of S to get a proof of 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 9. *$\vdash^L_{S,\varphi} \psi$ iff $\vdash^L_S \mathsf{AG}\varphi \to \psi$.*
Proof. *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 lenght 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* $\vdash^L_S \mathsf{AG}\varphi \to \psi$. ■

Now we can prove that the deductive machinery obtained by adding the locality axioms preserves translations of properties.

Theorem 10. *Given a translation* $\tau : L \to L'$, *if* $\vdash^L \varphi$, *then* $\vdash^L_{Loc(\tau)} \tau(\varphi)$.
Proof. *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: A13 and A18. For A18, 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 A13, 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:*

$$\vdash^{L'}_{Loc(\tau)} \tau(\langle\mathbf{U}\rangle\top \to \mathsf{A}\varphi \leftrightarrow [\mathbf{U}]\varphi).$$

Note the following property of $\mathsf{Done}()$*:*

$$\langle\alpha\rangle\top \to ((\mathsf{ANDone}(\alpha) \to \varphi) \leftrightarrow [\alpha]\varphi).$$

Using this property, we obtain that the sentence above is equivalent to:

$$\langle\tau(\mathbf{U})\rangle\top \to ([\tau(\mathbf{U})]\tau(\varphi) \leftrightarrow [\tau(\mathbf{U})]\tau(\varphi))$$

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 $\mathsf{Done}()$ *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* $\mathsf{Done}()$*, the left part is equivalent to:*

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

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

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

Using the fact that we have the following axiomatic schema in Loc(τ*):*

$$\langle\tau(\mathbf{U})\rangle\top \rightarrow \mathsf{AFDone}(\tau(\mathbf{U}))$$

and that from the formula 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:

$$\vdash^{L'}_{Loc(\tau)} \tau(\varphi) \to [\mathbf{U}]\tau(\varphi)$$

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:*

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

this is equivalent to:

$$\vdash^{L'}_{Loc(\tau)} \tau(\varphi) \to \neg\tau(\psi) \land (\langle\tau(\mathbf{U})\rangle\tau(\varphi) \lor [\tau(\mathbf{U})]\bot \to \tau(\varphi)).$$

It is not hard to prove that this formula implies $\tau(\varphi) \to (\neg\tau(\psi) \lor \mathsf{EN}\tau(\varphi))$*, 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.* ■

Recall that a theory presentation is a tuple $P = \langle L, A\rangle$ where L is a language and A is a set of axioms; it defines a theory, which is the set of consequences of the axioms. We say that $\vdash_P \varphi$ when $\vdash^L_A \varphi$. Later on we use the notion of theory presentation to define components. To that end, it is important to define interpretations between theory presentations.

Definition 11. *Given two theory presentations* $P = \langle L, A\rangle$ *and* $P' = \langle L', A'\rangle$*, then an interpretation is given by a translation* $\tau : L \to L'$ *such that* $\vdash_{P',Loc(\tau)} \tau(\varphi)$*, for every* $\varphi \in A$*.* □

As shown in [13], interpretations between theory presentations in structural logics preserve consequence; we can recast this result here if we add locality axioms to deductions. In this sense, it is important to understand that, from the logical point of view, we are considering a different deduction system per component. The resulting deduction system (of the final system) is determined by the way in which the different components are put together (as described below, a component is basically a theory presentation). First, let us prove that interpretations preserve consequences.

Theorem 11. *Given two theory presentations* $P = \langle L, A\rangle$ *and* $P' = \langle L', A'\rangle$ *and an interpretation* $\tau : P \to P'$*, we have that:*

$$\vdash_P \varphi \Rightarrow \vdash_{P',Loc(\tau)} \tau(\varphi)$$

Proof. *Suppose that we have* $\vdash_P \varphi$*; then we have a proof which uses some finite number of axioms of* P *(by the definition of proof). Let us say the proof is* $\varphi_1, \dots, \varphi_n$*; but then we have that:*

$$\vdash^L \mathsf{AG}\varphi_1 \land \dots \mathsf{AG}\varphi_n \to \varphi$$

by theorem 9, and then by theorem 10 we have that

$$\vdash^{L'}_{Loc(\tau)} \tau(\mathsf{AG}\varphi_1 \wedge \dots \mathsf{AG}\varphi_n \to \varphi).$$

But using the definition of translation we get:

$$\vdash^{L'}_{Loc(\tau)} \mathsf{AG}(\tau(\varphi_1)) \wedge \dots \wedge \mathsf{AG}(\tau(\varphi_n)) \to \tau(\varphi).$$

We know that $\vdash_{P',Loc(\tau)} \tau(\varphi_i)$ *for every* $0 \leq i \leq n$*, and therefore by generalization we obtain that* $\vdash_{P',Loc(\tau)} \mathsf{AG}(\tau(\varphi_i))$ *for every* $0 \leq i \leq n$*, and then using modus ponens we have that* $\vdash_{P',Loc(\tau)} \tau(\varphi)$*.* ■

3 Components and Violations.

In addition to the locality axioms, we need some other axioms to facilitate reasoning about violations. Violations, informally, occur when an obligation is not fulfilled (or a forbidden action is performed), and therefore it is natural to argue that executing allowed actions does not introduce more violations into the system. Later on in this paper, we define in detail the concept of module or component. For now we just say that a component C has a language L as described above; this language has a set of violations and therefore we can define a predicate V_L which, roughly speaking, defines which violations are true and which are false in the current state of the component (they define a state of violation). A state of violation then is a predicate $V_L = *\mathtt{v_1} \wedge \dots \wedge *\mathtt{v_n}$, where $\{\mathtt{v_1}, \dots, \mathtt{v_n}\} = V_0$ and $*$ is blank or $\neg$, i.e., this predicate describes a state where a subset of the violations are true and other violations are false. On the other hand, we can use the predicate $\mathbf{V}_C = \mathtt{v_1} \vee \dots \vee \mathtt{v_n}$ to detect if in a state some violation is true. Note that $\neg\mathbf{V}_C$ says that no violation is true, and therefore this predicate allows us to define normative states (and normative traces).

Obviously, we can build a lattice of violation states (see below) and we can abstract the states of the system using this lattice of violations by forming equivalence classes of states using the predicate V_C.

As explained above, one important requirement to ask for is that allowed actions must not introduce new violations. This is called the *GGG* (Green-Green-Green) condition in [20]; as the name of this condition indicates, from a “green” state (without violations), performing an allowed transition (a “green” transition), we must reach a “green” state. We take this principle further, and we say that, if from a state with a violation state V_C we perform an allowed action, then the state of violation is preserved or improved (i.e., it cannot happen that a violation, which is not true in a given state, becomes true after executing an allowed action). This principle can be formally specified by the following (finite) set of axioms:

- $\neg v_i \wedge \mathsf{P}^j(\alpha) \to [\alpha]\neg v_i$ for every violation v_i, action term α and permission index j.

Recall that the set of action terms modulo the set of axioms of boolean algebra is finite (the number of equivalence classes are finite). Given a component C, we call this set of axioms $G(C)$. From these axioms we can prove the GGG requirement.

Theorem 12. *For a given component C, $G(C) \vdash \neg\mathbf{V}_C \wedge \mathsf{P}(\alpha) \rightarrow [\alpha]\neg\mathbf{V}_C$.*

In the next section we define in detail the notion of components, and we show how we can put together different components to make a system.

3.1 Putting Together Deontic Specifications.

In this section we focus on the notion of component and the concepts needed to enable compositions of different components. First, we define the notion of "upgrading" and "degrading" formulae. These formulae are built from violation propositions; an upgrading formula says that one or more violations will become false in the future, and a degrading formula says that one or more violations will become true in the future. These kind of formulae are useful to analyze when, in a given state, a component will go into a worse violation state or will go into a better violation state (with perhaps an absence of violations). This is an important aspect when analyzing fault-tolerant software, as we want to know from which error states we cannot recover and from which ones we can make some improvement. An order between violation states is needed to formalize upgrading and degrading formulae. Note that, for every violation state (predicates $V_L = *\mathtt{v}_1 \wedge \cdots \wedge *\mathtt{v}_\mathtt{n}$ over a language L), we can define a set:

$$\mathcal{U}(V_L) = \{v_i \mid v_i \text{ appears without a negation in } V_L\}.$$

These sets induce an order $\leq_v$ over violation states as follows:

$$V_L \leq_v V'_L \Leftrightarrow U(V_L) \supseteq U(V'_L)$$

Note that $\leq_v$ is contravariant with respect to $\subseteq$. Intuitively, only the violations which are true at that point appear in $\mathcal{U}(V_L)$. The relationship $\leq_v$ denotes a relationship of "improvement", the set of sets $U(V_L)$ form a lattice; we study this fact in detail later on. We denote by $<_v$ the strict version of $\leq_v$.

Given a component (see below), an upgrading formula in this component is a formula which specifies some way of recovering from a violation, i.e., an upgrading formula identifies a recovery action for a given violation. Taking into account the order defined over the violation states, we can say that a recovery action improves the state of violation of a component. These facts inspire the following definition.

Definition 12. *Given a language L, the set of upgrading formulae for L is defined as follows:*

- *If V_L and V'_L are violation states in L, φ is a formulae in L and $V_L <_v V'_L$ then,*

$$(V_L \to ([\alpha_1; \dots ; \alpha_n]V'_L) \wedge (\langle \alpha_1; \dots ; \alpha_n \rangle \top))$$

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

$\square$

Here we define $[\alpha; \beta]\varphi = [\alpha][\beta]\varphi$. Also note that we require that there must exist some way of executing the sequence of actions in the upgrading formulae (otherwise an impossible action will be a recovery action!). One might think that the condition $(V \to \langle \alpha_1; \dots ; \alpha_n \rangle V'_L)$ is a better formalization of upgrading formulae, but it is too weak; it says that, sometimes, by executing $\alpha_1, \dots, \alpha_n$ we eliminate some violations, and it does not give us any details about those scenarios corresponding to bad situations from we can effectively recover. Instead, using box modalities, we can ensure that executing the sequence of actions in the predicate we can always recover from a violation state. At first sight, this could be too strong a requirement, however, we can refine the sequence of actions $\alpha_1; \dots ; \alpha_n$ as much as required to describe exactly the actions needed to upgrade the actual state of violations.

In the same way, we can define the set of degrading formulae, which intuitively define actions which introduce violations.

Definition 13. *Given a language L, the set of degrading formulae is defined as follows:*

- *If V_L and V'_L are violation states in L, φ is a formula in L and $V'_L <_v V_L$ then,*

$$(V_L \to ([\alpha_1; \dots ; \alpha_n]V'_L) \wedge \langle \alpha_1; \dots ; \alpha_n \rangle \top))$$

is a degrading formula, where $\alpha_1, \dots, \alpha_n$ are actions in the language L.

$\square$

A component is a piece of specification which is made up of a language, a finite set of axioms, and a set of additional axioms which formalize implicit assumptions on the components (e.g., locality axioms). These implicit axioms are not intended to be defined by a designer; instead, they are automatically obtained from the structure of our system (using the relationships between the different components).

Definition 14. *A component is a tuple $\langle L, A, S \rangle$ where:*

- *L is a language as described in earlier sections.*

- *A is a finite set of axioms (the axioms given by the designers).*
- *S is a set of axioms (the system axioms).*

$\square$

Given a component $C = \langle L, A, S \rangle$ we denote by $\vdash_C \varphi$ the assertion $\vdash^L_{A,S} \varphi$. Usually we consider that $G(L) \subseteq S$ (i.e., the GGG predicate is in the system axioms). A mapping between two components is basically an interpretation between the theory presentations that define them.

Definition 15. *A mapping $\tau : C \to C'$ between two components $C = \langle L, A, S \rangle$ and $C' = \langle L', A', S' \rangle$ is a translation $\tau : L \to L'$ such that:*

- *$\vdash_{C'} \tau(\varphi)$, for every $\varphi \in A \cup S$.*
- *$\vdash_{C'} Loc(\tau)$.*

$\square$

It is worth noting that we require that the locality axioms must be theorems in the target component to ensure that the properties of the smaller component are preserved (as proved by theorem 11). This is expressed by the following corollary.

Corollary 2. *If $\tau : C \to C'$ 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 [4], where concepts coming from category theory are used to put together components of a specification. The same ideas are used in [11] and [12], 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, note that the collection of all the languages and all the translations between them form 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 (which straightforwardly satisfies associativity). Components and mappings between them also constitute a category.

Theorem 13. *The collection of all components* **Comp** *and all the arrows between them form the category* ***Comp****.*
Proof. *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 $\vdash_C Loc(id_C)$ (where id_C is the identity tranlation). And, if we have tranlations $\tau : C_1 \to C_2$ and $\tau' : C_2 \to C_1$, then $\vdash_{C_3} Loc(\tau' \circ \tau)$. The proofs are straightforward using the properties of the logic.* ■

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 finite 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 [14] and [13]).

Theorem 14. *The category* ***Comp*** *is finitely cocomplete.*
Proof. *We prove that the functor $Sign$: **Comp** $\to$ **Sign** reflects limits, and since **Sign** is finitely cocomplete, hence **Comp** is finitely cocomplete too.*

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

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

*is a colimit object in **Comp**. For each component C_i, the translation to C is given by α_i. (Note that it is injective on actions, since α_i is injective on actions.) 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 $\vdash^L_{\bigcup_{i \in I} \alpha_i(Loc(C_i))} \alpha_i(Loc(C_i))$ and $\vdash^L_{\bigcup_{i \in I} \alpha_i(G(L_i))} \alpha_i(G(L_i))$, and by theorem 11 we have that*

$$\vdash_\Gamma \alpha_i(\varphi)$$

*where $\Gamma = \bigcup_{i \in I} \alpha_i(A_i) \cup \bigcup_{i \in I} \alpha_i(Loc(C_i)) \cup \bigcup_{i \in I} \alpha_i(S_i)$, 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 $\langle C', \beta : C_i \to C' \rangle$, then in **Sign** we have an unique morphism $\psi : L \to L'$ (where L' is the language of C'). It is straightforward to check that ψ can be extended to an unique $\psi : C \to C'$. This finishes the proof.* ■

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 next section we exhibit an example. First, we describe how the lattice of violations of a system can be approximated from the laticce of violations of each of its components.

4 Calculating Violations.

In each component, in a given state of execution of that component, we have a set of violation predicates which are valid. This set of violation predicates can be illustrated as a partially oredered set (using a classic graphical illustration of partially ordered sets). Having a visual representation of how the violations in an execution of a program behave is useful to analyze specifications to determine what can go wrong and what to do to fix it. Each state of violation can be thought of as the set of violations which are true at that state; then the inclusions between these sets give us a diagram of degrading, whereas the opposite arrows of inclusion give us an upgrading diagram of violations. For each set of violation predicates we can calculate a corresponding diagram of sets of violations and inclusions (or opposites of inclusions), which form a category. The important point is that these diagrams illustrate how violation states are related with respect to upgrading and degrading actions and, furthermore, given a collection of violation diagrams, a colimit of them gives us a good approximation to the violation diagram of the system obtained when the components are put together, when some conditions are satisfied.

As explained above, we consider a set $\mathcal{V} = \{\mathtt{v}_1, \mathtt{v}_2, \mathtt{v}_3 \dots\}$ of violations. Then we can define the small category $\mathcal{C}(\mathcal{V})$ which has as objects subsets of $\mathcal{V}$ and as arrows functions between these sets. We want a particular part of this category which corresponds to upgrading and degrading actions. First, consider the category **Pos** whose objects are partially ordered sets (which are categories) and whose morphism are functors between them (order preserving mappings). This category is complete and cocomplete. We call a functor $F : I \to \mathcal{C}(V)$ a *degrading diagram*, where I is a preordered set such that to each arrow $i \to j$ between two elements of I, F maps it to an inclusion $F(i) \hookrightarrow F(j)$ in $\mathcal{C}(V)$. Now, a morphism $G : D \to D'$ between two degrading diagrams $D : I \to \mathcal{C}(V)$ and $D' : J \to \mathcal{C}(V)$ is a functor (an order preserving mapping) $F : I \to J$ between I and J and a natural transformation $\alpha : D \to D'F$. Naturality means that the following diagram commutes:

$$\begin{tikzcd}
i & D(i) \arrow[r, "\alpha_i"] & D'F(i) \\
j \arrow[u] & D(j) \arrow[u, hook] \arrow[r, "\alpha_j"] & D'F(j) \arrow[u, hook]
\end{tikzcd}$$

The category **Deg** is the category whose objects are violation diagrams and whose arrows are pairs $\langle F : I \to J, \alpha : D \to D'F \rangle : D \to D'$ as explained above. Since the category **Pos** and the category $\mathcal{C}(V)$ are finitely cocomplete, then for **Deg** colimits can be calculated pointwise (see [17]); therefore **Deg** is finitely cocomplete.

Theorem 15. *The category* **Deg** *is finitely cocomplete.*

On the other hand, an *upgrading diagram* is a functor $F : I^{op} \to \mathcal{C}(V)$, where I is a preorder. Note that we take the dual of this category since upgrading diagrams are contravariant with respect to inclusion. However, the opposite of partially ordered set is a partially ordered set; therefore, an upgrading diagram is essentially the same as a degrading diagram, but we draw the arrows in the other direction (see the example below). A morphism between two upgrading diagrams $D : I^{op} \to \mathcal{C}(V)$ and $D' : J^{op} \to \mathcal{C}(V)$ is a tuple $\langle F : I^{op} \to J^{op}, \alpha : D \to D'F\rangle$. Since the dual of a partially ordered set is a partially ordered set, we have the following property:

Theorem 16. *The category* ***Upg*** *is finitely cocomplete.*

To illustrate the idea of how colimits are built over violation diagrams, consider the degrading diagrams of figure 1 Note that diagrams D_2 and D_3

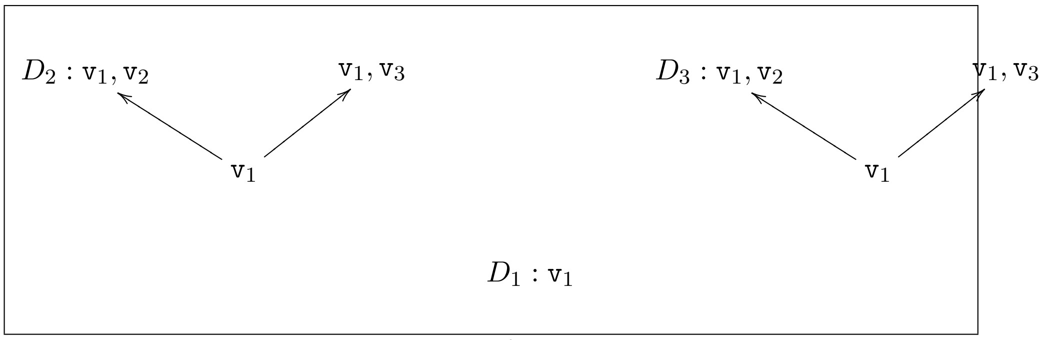
$D_2 : \mathtt{v}_1, \mathtt{v}_2$
$\mathtt{v}_1, \mathtt{v}_3$
$\mathtt{v}_1$
$D_3 : \mathtt{v}_1, \mathtt{v}_2$
$\mathtt{v}_1, \mathtt{v}_3$
$\mathtt{v}_1$
$D_1 : \mathtt{v}_1$

Figure 1: Examples of degrading diagrams

are isomorphic, while the diagram D_1 only has $\{\mathtt{v}_1\}$ (i.e., it only has an isolated point). If we consider the following morphisms between degrading diagrams (recall that they are made up of a natural transformation and a functor): $F_1 : D_1 \to D_2$ and $F_2 : D_1 \to D_3$, which map $\{\mathtt{v}_1\}$ to the the same set in each diagram, then the pushout object obtained $(D_2 +_{D_1} D_3)$ from the above violation diagrams is shown in figure 2. In other words, colimits

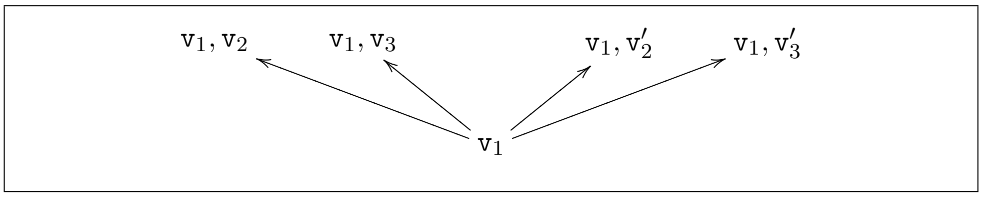
$\mathtt{v}_1, \mathtt{v}_2$
$\mathtt{v}_1, \mathtt{v}_3$
$\mathtt{v}_1, \mathtt{v}'_2$
$\mathtt{v}_1, \mathtt{v}'_3$
$\mathtt{v}_1$

Figure 2: Degrading diagram $C_2 +_{C_1} C_3$

join the common parts indicated by the given diagram and they separate the other parts (creating new names or violation predicates to distinguish between unrelated propositions). The same applies to upgrading diagrams.

Given a component $C = \langle L, A, S\rangle$, we can define its degrading and upgrading diagrams formally as follows. The degrading diagram is a functor $D_C : I_C \to \mathcal{C}(V)$, where the elements of I_C are defined as follows. If V, V' are two violation states of C and $\vdash_C V \to ([\alpha_1; \dots; \alpha_n]V') \wedge (\langle \alpha_1; \dots \alpha_n\rangle \top)$ is a degrading 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 $D_I : I_C \to \mathcal{C}(V)$ is defined as follows.

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

Similarly, we define an upgrading diagram $U_C : J_C^{op} \to \mathcal{C}(V)$; in this case we draw the arrows following the direction in J_C^{op}.

Note that though the potential upgrading and degrading predicates in a component are possibly infinite, the diagram is finite, since we have a finite number of violation states, which implies that we have equivalence classes of degrading and upgrading functions that represent the same transition between two violation state.

We present a simple example to illustrate these notions (a more complex example is given in section 5). Consider the following scenario. We have a system where we have two coolers which must maintain the low temperature of a processor. We specify the coolers as two instances of the specification shown in figure 3: In this specification we have actions: `on` (to turn on the

C1.$\mathsf{B} \to \neg\mathtt{v} \wedge \neg\mathtt{on} \wedge \neg\mathtt{high}$	**C11**.$\mathtt{v} \to [\overline{on}]\mathtt{v}$
C2.$\neg\mathtt{on} \to [\overline{\mathtt{on}}]\neg\mathtt{on}$	**C12**.$[\mathtt{ghigh}]\mathtt{high}$
C3.$[\mathtt{off}]\neg\mathtt{on}$	**C13**.$\neg\mathtt{high} \to [\overline{ghigh}]\neg\mathtt{high}$
C4.$\mathtt{on} \to [\overline{\mathtt{off}}]\mathtt{on}$	**C14**.$[\mathtt{on}]\neg\mathtt{high}$
C5.$[\mathtt{on}]\mathtt{on}$	**C15**.$\mathtt{high} \to [\overline{on}]\mathtt{high}$
C6.$\neg\mathtt{high} \to \mathsf{P}(\mathbf{U})$	**C16**.$\langle\mathtt{ghigh}\rangle\top$
C7.$\mathtt{high} \to \mathsf{O}(\mathtt{on})$	**C17**.$\langle\mathtt{off}\rangle\top$
C8.$\mathsf{F}(\overline{on}) \to [\overline{on}]\mathtt{v}$	**C18**.$\langle\mathtt{on}\rangle\top$
C9.$\mathtt{v} \to [\mathtt{on}]\neg v$	**C19**.$\mathtt{off} \sqcap \mathtt{on} =_{act} \emptyset$
C10.$\mathsf{P}(\mathtt{on})$	

Figure 3: Specification of a cooler

cooler), `ghigh` (this action indicates when the temperature of the cooler is high) and `off` (this action turns the cooler off). Most of the axioms specify the behavior of these actions, we can highlight the following axioms. Axiom **C6** says that, if the temperature is low, then any action is allowed. Axiom

C7 says that, if the temperature is high, then the cooler ought to be on. Axioms **C8** indicates when a violation arises. On the other hand, axiom **C9** says that `on` is a recovery action for violation `v`. Axioms **C15-C17** say that the actions `ghigh`, `on` and `off` can always be executed. Finally, axiom **C18** says that actions `on` and `off` are disjoint, and axiom **C10** says that the action `on` is always allowed (which implies that it never causes a violation).

Now suppose that, perhaps since we want some redundancy in the system, we are interested in having two coolers, an specification of two coolers can be obtained taking the coproduct **C**+**C**. In this case, different variables, actions and deontic predicates are created to distinguish the two instances of the same specification. see figure 4. Let us investigate the degrading

$\mathbf{C1_i}.\mathsf{B} \to \neg \mathtt{v}_i \wedge \neg \mathtt{on}_i \wedge \neg \mathtt{high}_i$	$\mathbf{C11_i}.\mathtt{v}_i \to [\overline{\mathtt{on}_i}]\mathtt{v}_i$
$\mathbf{C2_i}.\neg \mathtt{on}_i \to [\overline{\mathtt{on}_i}]\neg \mathtt{on}_i$	$\mathbf{C12_i}.[\mathtt{ghigh}_i]\mathtt{high}_i$
$\mathbf{C3_i}.[\mathtt{off}_i]\neg \mathtt{on}_i$	$\mathbf{C13_i}.\neg \mathtt{high}_i \to [\overline{\mathtt{ghigh}_i}]\neg \mathtt{high}_i$
$\mathbf{C4_i}.\mathtt{on}_i \to [\overline{\mathtt{off}_i}]\mathtt{on}_i$	$\mathbf{C14_i}.[\mathtt{on}_i]\neg \mathtt{high}_i$
$\mathbf{C5_i}.[\mathtt{on}_i]\mathtt{on}_i$	$\mathbf{C15_i}.\mathtt{high}_i \to [\overline{on_i}]\mathtt{high}_i$
$\mathbf{C6_i}.\neg \mathtt{high}_i \to \mathsf{P}^i(\mathbf{U})$	$\mathbf{C16_i}.\langle \mathtt{ghigh}_i \rangle \top$
$\mathbf{C7_i}.\mathtt{high}_i \to \mathsf{O}^i(\mathtt{on}_i)$	$\mathbf{C17_i}.\langle \mathtt{off}_i \rangle \top$
$\mathbf{C8_i}.\mathsf{F}(\overline{on_i}) \to [\overline{on_i}]\mathtt{v}_i$	$\mathbf{C18_i}.\langle \mathtt{on}_i \rangle \top$
$\mathbf{C9_i}.\mathtt{v}_i \to [\mathtt{on}_i]\neg v_i$	$\mathbf{C19_i}.\mathtt{off}_i \sqcap \mathtt{on}_i =_{act} \emptyset$
$\mathbf{C10_i}.\mathsf{P}^i(\mathtt{on}_i)$	where $i = 1, 2$

Figure 4: Coproduct **C** + **C**

diagram of the component **C**. First, we can prove $\vdash_{\mathbf{C}} \neg \mathtt{v} \to [\mathtt{ghigh}; \overline{\mathtt{on}}]\mathtt{v}$.

1.	$\vdash_{\mathbf{C}} [\mathtt{ghigh}]\mathtt{high}$	**C11**
2.	$\vdash_{\mathbf{C}} \mathtt{high} \to \mathsf{O}(\mathtt{on})$	**C7**
3.	$\vdash_{\mathbf{C}} \mathsf{O}(\overline{\mathtt{on}}) \to \mathsf{F}(\overline{\mathtt{on}})$	Def.O()
4.	$\vdash_{\mathbf{C}} \mathsf{F}(\overline{\mathtt{on}}) \to [\overline{\mathtt{on}}]\mathtt{v}$	**C5**
5.	$\vdash_{\mathbf{C}} \mathtt{high} \to [\overline{\mathtt{on}}]\mathtt{v}$	PL,2,3,4
6.	$\vdash_{\mathbf{C}} [\mathtt{ghigh}][\overline{\mathtt{on}}]\mathtt{v}$	ML, 5, 1
7.	$\vdash_{\mathbf{C}} [\mathtt{ghigh}; \overline{\mathtt{on}}]\mathtt{v}$	Def.;

That is, the degrading diagram of specification **C** is the diagram (a) shown in figure 5. Meanwhile, the degrading diagram of the coproduct **C** + **C** is shown in figure 5 (b). To prove that this is correct, we have to prove that there exists action expressions α_1, α_2, α_3 and α_4 which make the following statements true:

1. $\vdash_{\mathbf{C+C}} \neg \mathtt{v}_1 \wedge \neg \mathtt{v}_2 \to ([\alpha_1](\mathtt{v}_1 \wedge \neg \mathtt{v}_2) \wedge \langle \alpha_1 \rangle \top)$.

2. $\vdash_{\mathbf{C+C}} \neg \mathtt{v}_1 \wedge \mathtt{v}_2 \to ([\alpha_2](\neg v_1 \wedge \neg v_2) \wedge \langle \alpha_2 \rangle \top)$.

3. $\vdash_{\mathbf{C+C}} \mathtt{v}_1 \wedge \neg \mathtt{v}_2 \to ([\alpha_3](\mathtt{v}_1 \wedge \mathtt{v}_2) \wedge \langle \alpha_3 \rangle \top)$.

(a)

$$\begin{tikzcd} \mathtt{v} \\ \emptyset \arrow[u] \end{tikzcd}$$

(b)

$$\begin{tikzcd} & \mathtt{v}_1, \mathtt{v}_2 & \\ \mathtt{v}_1 \arrow[ur] & & \mathtt{v}_2 \arrow[ul] \\ & \emptyset \arrow[ul] \arrow[ur] & \end{tikzcd}$$

Figure 5: Degrading diagrams of **C** and **C** + **C**

4. $\vdash_{\mathbf{C+C}} \neg\mathtt{v}_1 \wedge \mathtt{v}_2 \to ([\alpha_4](\mathtt{v}_1 \wedge \mathtt{v}_2) \wedge \langle\alpha_3\rangle\top)$.

For example, for item 1 we can prove:

$$\vdash_{\mathbf{C+C}} \neg\mathtt{v}_1 \wedge \neg\mathtt{v}_2 \to [\mathtt{ghigh}_1; \mathtt{ghigh}_1 \sqcup \mathtt{off}_1]\neg\mathtt{v}_2 \wedge \mathtt{v}_1$$

as follows:

1.	$\vdash_{\mathbf{C}} \neg\mathtt{v} \to [\mathtt{ghigh}; \overline{\mathtt{on}}]\mathtt{v}$	See above.
2.	$\vdash_{\mathbf{C+C}} \tau_1(\neg\mathtt{v}) \to [\tau_1(\mathtt{ghigh}); \tau_1(\overline{\mathtt{on}})]\tau_1(\mathtt{v})$	Theorem 11
3.	$\vdash_{\mathbf{C+C}} \neg\mathtt{v}_1 \to [\mathtt{ghigh}_1; \mathtt{ghigh}_1 \sqcup \mathtt{off}_1]\mathtt{v}_1$	Def.τ_1
4.	$\vdash_{\mathbf{C+C}} \neg\mathtt{v}_2 \to [\mathtt{ghigh} \sqcup \mathtt{off}_1 \sqcup \mathtt{on}_1]\neg\mathtt{v}_2$	Loc.
5.	$\vdash_{\mathbf{C+C}} \neg\mathtt{v}_2 \to [\mathtt{ghigh}_1]\neg\mathtt{v}_2$	DPL 4
6.	$\vdash_{\mathbf{C+C}} \neg\mathtt{v}_2 \to [\mathtt{ghigh}_1 \sqcup \mathtt{off}_1]\neg\mathtt{v}_2$	DPL 4
7.	$\vdash_{\mathbf{C+C}} \neg\mathtt{v}_2 \to [\mathtt{ghigh}_1; \mathtt{ghigh}_1 \sqcup \mathtt{off}_1]\neg\mathtt{v}_2$	DPL, 5, 6
8.	$\vdash_{\mathbf{C+C}} \neg\mathtt{v}_1 \wedge \neg\mathtt{v}_2 \to [\mathtt{ghigh}_1; \mathtt{ghigh}_1 \sqcup \mathtt{off}_1]\neg\mathtt{v}_2 \wedge \mathtt{v}_1$	DPL, 4, 7

Note that, in this proof, we have used the theorem proven in component **C** to prove the statement in the specification **C** + **C**; this shows the way in which the theorems proven in the components can be reused to prove properties about the entire system. We can also prove: $\vdash_{\mathbf{C+C}} \neg\mathtt{v}_1 \wedge \neg\mathtt{v}_2 \to \langle\mathtt{ghigh}_1; \mathtt{ghigh}_1 \sqcup \mathtt{off}_1\rangle\top$, as follows:

1.	$\vdash_{\mathbf{C+C}} \langle\mathtt{ghigh}_1\rangle\top$	$\mathbf{C16_1}$
2.	$\vdash_{\mathbf{C+C}} \langle\mathtt{off}_1\rangle\top$	$\mathbf{C17}_1$
3.	$\vdash_{\mathbf{C+C}} \langle\mathtt{ghigh}_1 \sqcup \mathtt{off}_1\rangle\top$	DPL, 1, 2
4.	$\vdash_{\mathbf{C+C}} [\mathtt{ghigh}_1]\langle\mathtt{ghigh}_1 \sqcup \mathtt{off}_1\rangle\top$	**GN**, 3
5.	$\vdash_{\mathbf{C+C}} \langle\mathtt{ghigh}_1\rangle\langle\mathtt{ghigh}_1 \sqcup \mathtt{off}_1\rangle\top$	DPL, 3, 4
6.	$\vdash_{\mathbf{C+C}} \langle\mathtt{ghigh}_1; \mathtt{ghigh}_1 \sqcup \mathtt{off}_1\rangle\top$	Def.;

and therefore we obtain the arrow $\emptyset \to \mathtt{v}_1$ in the degrading diagram. In the same way, the remaining items in the list can be proven, and therefore, we obtain the degrading diagram of figure 5 (b).

In practice we can coordinate the different instances of the specification via some actions. However, in this case (depending in which actions or variables we coordinate the specifications) the indenpendence of the degrading and upgrading diagrams of each component might not be preserved when putting the components together. An obvious example is to coordinate the two coolers via actions `ghigh` and `on` (i.e., they have the same sensor and the same button turning on both sensors). In this case, both sensors go into violation at the same time. But note that, if we coordinate the two coolers only for action `on`, then the indendence of the degrading or upgrading diagram is preserved. This can be proven using the *GGG* condition and the fact that the action `on` is always allowed in both components, i.e., this action never introduces a violation.

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 $D : I \to \mathbf{Comp}$ with components $C_1, \dots, C_n$, and colimit $\langle C, \tau_i : C_i \to C\rangle$, 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 $\tau_i(p_i) = p = \tau(p_j)$.

Our first theorem says that, when we have two components which do not coordinate via any action, then the degrading diagrams of each component are respected by the degrading diagram of the specification obtained when we put both components together.

Theorem 17. *Consider two components C_1 and C_2, with degrading diagrams $D_{C_1} : I_{C_1} \to \mathcal{C}(V)$ and $D_{C_2} : I_{C_2} \to \mathcal{C}(V)$, respectively. Let $D_{C_1+C_2} : I_{C_1+C_2} \to \mathcal{C}(V)$ be the degrading diagram of C_1+C_2 (the coproduct of C_1 and C_2), then there is a morphism $\langle F, \alpha\rangle : D_{C_1} + D_{C_2} \to D_{C_1+C_2}$, such that all the components of α are iso and F is faithful.*

Proof. *We prove that, if we have an arrow $V \to V'$ in the coproduct of the violation diagrams, then we have a degrading (or upgrading) action in the coproduct of the components, which identifies this arrow. From here, we can use the identities to map $D_{C_1} + D_{C_2}$ to $D_{C_1+C_2}$. Suppose that we have $V \to V'$ in $D_{C_1} + D_{C_2}$, then, by properties of coproducts, this arrow belongs to D_1 or D_2; if $V \to V'$ belongs to D_1, then we have that $\vdash_{C_1} (V \to [\alpha_1; \dots \alpha_n]V' \wedge \langle \alpha_1; \dots ; \alpha_n\rangle\top$, but then we have:*

$$\vdash_{C_1+C_2} \tau_1(V) \to [\tau_1(\alpha_1); \dots ; \tau_1(\alpha_n)]\tau_1(V') \wedge \langle \tau_1(\alpha_1); \dots \tau_1(\alpha_n)\rangle\top.$$

But note that $\tau(V)$ is not necessarily a violation state of $C_1 + C_2$, since the violations of C_2 are not considered there. But since C_1 and C_2 do not coordinate via any action, we know that $\tau_1(V) \wedge \tau_2(\neg \boldsymbol{V}_{C_2})$, is a violation *state of $C_1 + C_2$. From this, using the properties of locality, we obtain:*

$$\vdash_{C_1+C_2} \tau_1(V) \wedge \neg\tau_2(V_{C_2}) \to \langle \tau_1(\alpha_1) \sqcap \overline{\tau_2(\mathbf{U})}; \dots; \tau_1(\alpha_n) \sqcap \overline{\tau_2(\mathbf{U})} \rangle \tau_1(V') \wedge \neg\tau_2(\boldsymbol{V}_{C_2})$$

and

$$\vdash_{C_1+C_2} \tau_1(V) \wedge \neg\tau_2(V_{C_2}) \to [\tau_1(\alpha_1) \sqcap \overline{\tau_2(\mathbf{U})}; \dots; \tau_1(\alpha_n) \sqcap \overline{\tau_2(\mathbf{U})}] \tau_1(V') \wedge \neg\tau_2(\boldsymbol{V}_{C_2})$$

and therefore the degrading transition $V \to V'$ belongs to the degrading diagram of $C_1 + C_2$. ■

It is important to analyze in detail what this theorem says. If we have two components and we put them together without coordinating them via any action (they are totally disjoint), then the violation state of one component does not affect the other component and viceversa. The isomorhism of the components of the natural transformation indicate that the number of violations is preserved in each violation state of the components, and the faithfullness of the functor indicates that the "shape" of the degrading or upgrading diagrams of each component is preserved by the degrading diagram of the entire system. Obviously, this result can be extended for a more general setting, where we have many components which do not coordinate via any actions.

We can generalize this result to situations where the components with violations do not coordinate via any action (see below the example of the diarrheic philosophers where philosophers do not coordinate via any actions; they only coordinate with forks but forks, do not have violations).

Theorem 18. *Given a finite diagram $D : I \to$ **Comp** with components $C_1, \dots, C_n$, let $\langle C, \tau_i : C_i \to C \rangle$ be a colimit of D. If $C_{i_1}, \dots, C_{i_k}$ are all the components with violations in the language, and let $D_{i_1}, \dots, D_{i_k}$ be their degrading diagrams. If components $C_{i_1}, \dots, C_{i_k}$ do not coordinate via any action nor variable, then there is a morphism $\langle F, \alpha \rangle : D_{i_1} + \dots + D_{i_k} \to D_C$ where the components of α are iso and F is faithfull.*
Proof. *The diagram $D_{i_1} + \dots D_{i_k}$ is the coproduct of the diagrams $D_{i_1}, \dots, D_{i_k}$; we show that each arrow $V \to V'$ is also an arrow of D_C, and therefore, the morphism between $D_{i_1} + \dots + D_{i_k}$ and D_C is given by identities.*

Let $V \to V'$ be an arrow in some D_{i_j}; then we have:

$$\vdash_{C_{i_j}} V \to [\alpha_1; \dots; \alpha_n]V' \wedge \langle \alpha_1; \dots; \alpha_n \rangle \top.$$

Now , we have that:

$$\vdash_C \tau_{i_j}(V) \to [\tau_{i_j}(\alpha_1); \dots; \tau_{i_j}(\alpha_n)]\tau_{i_j}(V') \wedge \langle \tau_{i_j}(\alpha_1); \dots; \tau_{i_j}(\alpha_n) \rangle \top$$

Note that $\tau_{i_j}(V)$ is not necessarily a violation state in C. Let $v_1, \dots, v_n$ be the violations which do not appear in V (these are translations of violations *in other components), then $\tau_{i_j} \wedge \neg v_1 \wedge \cdots \wedge \neg v_m$ is a violation state of C. Let $a_1, \dots, a_q$ be the actions which are translations of actions of C_{i_j}. Since C_{i_j} does not coordinate via any action nor variable with the other of components with violations, by locality we have:*

$$\vdash_C \neg v_1 \wedge \cdots \wedge \neg v_m \to [\overline{a_1} \sqcap \cdots \sqcap \overline{a_q}] \neg v_1 \wedge \cdots \wedge \neg v_m$$

and therefore by properties of DPL we have:

$$\vdash_C \tau_{i_j}(V) \wedge \neg v_1 \wedge \cdots \wedge \neg v_m \to [\tau_{i_j}(\alpha_1) \sqcap \overline{a_1}; \dots; \tau_{i_j}(\alpha_n); \overline{a_q}] V' \wedge \neg v_1 \wedge \cdots \wedge \neg v_m$$

and by the independence axioms we have:

$$\vdash \tau_{i_j}(V) \wedge \neg v_1 \wedge \cdots \wedge \neg v_m \to \langle \tau_{i_j}(\alpha_1) \sqcap \overline{a_1}; \dots; \tau_{i_j}(\alpha_n); \overline{a_q} \rangle$$

which shows that we have an arrow $V \to V'$ in D_C. The result follows. ■

Note that the theorems above are also valid for upgrading diagrams; we only need to change the direction of the arrows in the proofs.

However, in practice components usually coordinate via some actions, and therefore it is important to have some result which can be applied to wider cases where components interact in some way. Note that the *GGG* predicate says that an execution of an allowed action cannot introduce a violation into a violation state. Then, if we coordinate two components on actions which are always allowed (i.e., they are safe), we can ensure that no violations are introduced when we execute a recovery (or a degrading) 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 $\{1, \dots, n\}$ are the permission indexes of L, and we say that α is safe in a component C if $\vdash_C \mathbf{P}(\alpha)$.

Theorem 19. *Given a diagram $C_1 \leftarrow C \to C_2$, and the pushout of this diagram, denoted by $C_1 +_C C_2$, if the actions in C, say $c_1, \dots, c_n$, are safe (i.e., $\vdash_C \mathbf{P}(c_i)$ for every i) and C, then there is a morphism $\langle F, \alpha \rangle : U_{C_1} + U_{C_2} \to U_{C_1 +_C C_2}$, such that all the components of α are iso and F is faithful.*
Proof. *The proof is similar to the proof of theorem 17. Suppose that we have an upgrading transition $V \to V'$ in $U_{C_1} + U_{C_2}$. For the case that $V \to V'$ belongs to U_{C_1}, we proceed as follows: let $a_1, \dots, a_k$ be the primitive actions of C_1 and $b_1, \dots, b_m$ be the primitive actions of C_2. Let us use $C_1 - C_2$ for the expression:*

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

and similarly for $C_2 - C_1$. We have that:

$$\vdash_{C_1 +_C C_2} \tau_1(V) \to [\tau_1(\alpha_1); \dots; \tau_1(\alpha_n)] \tau(V') \wedge \langle \tau_1(\alpha_1); \dots; \tau_1(\alpha_n) \rangle \top.$$

Note that $\tau_1(V)$ and $\tau_1(V')$ are not necessarily violation states of $C_1 +_C C_2$. Now let $v_1, \ldots, v_t$ 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:

$$\vdash_{C_1 +_C C_2} \neg v_1 \wedge \ldots \neg v_t \rightarrow [\overline{\tau_2(\mathbf{U})}]\neg v_1 \wedge \cdots \wedge \neg v_t.$$

Also we know that:

$$\vdash_{C_1 +_C C_2} \neg v_1 \wedge \ldots \neg v_t \rightarrow [c_1 \sqcup \cdots \sqcup c_n]\neg v_1 \wedge \cdots \wedge \neg v_t.$$

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

$$\vdash_{C_1 +_C C_2} \neg v_1 \wedge \ldots \neg v_t \rightarrow [(c_1 \sqcup \cdots \sqcup c_n) \sqcup \overline{\tau_2(\mathbf{U})}]\neg v_1 \wedge \cdots \wedge \neg v_t$$

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

$$\vdash_{C_1 +_C C_2} \tau_1(V) \wedge \neg v_1 \wedge \cdots \wedge \neg v_t \rightarrow [\alpha_1 \sqcap \overline{C_2 - C_1}; \ldots; \alpha_n \sqcap \overline{C_2 - C_1}]\tau_1(V') \wedge \neg v_1 \wedge \cdots \wedge \neg v_t$$

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. ■

This theorem can be expressed by means of a slogan:

Coordination on safe actions is safe.

This property can be generalized when we have a finite number of components and they only interact (or coordinate) by means of safe actions. Note that in the theorem we require that component C does not have any violations, i.e., in other words, 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 thesis. 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.

5 Revisiting the Diarrheic Philosophers.

Now, we show an example to illustrate the application of these theorems in practice. We revisit the example of the diarrheic philosophers (which was introduced without the notion of components in [7, 6]). Here we follow the main ideas introduced in [12] to modularize the design; note that the design obtained by modularization is clearer that the original one. First, let us consider the specification of a fork. The actions of a fork are:

$$\{\texttt{l.up}, \texttt{l.down}, \texttt{r.up}, \texttt{r.down}\}$$

and the predicates are

$$\{\texttt{l.up?}, \texttt{r.up?}\}$$

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), this allows us to use theorem 18 to prove that the recovery of one philosopher does not cause any violations in the other philosophers. The axioms of the fork are shown in figure 6. The axioms specify the behavior of a fork. As explained above,

XFork:

$\mathbf{f_1}.\mathsf{B} \rightarrow \neg\texttt{l.up?} \wedge \neg\texttt{r.up?}$	$\mathbf{f_8}.[\texttt{r.up}]\texttt{r.up?}$
$\mathbf{f_2}.[\texttt{l.up}]\neg\texttt{l.up?}$	$\mathbf{f_9}.[\texttt{r.down}]\texttt{r.down?}$
$\mathbf{f_3}.[\texttt{l.down}]\neg\texttt{l.down?}$	$\mathbf{f_{10}}.\neg\texttt{r.up?} \rightarrow [\overline{\texttt{r.up}}]\neg\texttt{r.up?}$
$\mathbf{f_4}.\neg\texttt{l.up?} \rightarrow [\overline{\texttt{l.up}}]\neg\texttt{l.up?}$	$\mathbf{f_{11}}.\texttt{r.up?} \rightarrow [\overline{\texttt{r.up}}]\texttt{r.up?}$
$\mathbf{f_5}.\texttt{l.up?} \rightarrow [\overline{\texttt{l.down}}]\texttt{l.up?}$	$\mathbf{f_{12}}.\neg\texttt{r.up?} \rightarrow [\texttt{r.down}]\bot$
$\mathbf{f_6}.\neg(\texttt{l.up?} \wedge \texttt{r.up?})$	$\mathbf{f_1 4}.\texttt{l.up?} \rightarrow \langle\texttt{l.down}\rangle\top$
$\mathbf{f_7}.\neg\texttt{l.up?} \rightarrow [\texttt{l.down}]\bot$	$\mathbf{f_1 5}.\texttt{r.up?} \rightarrow \langle\texttt{r.down}\rangle\top$

Figure 6: XFork specification

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 $\mathbf{f_6}$), meaning that only one of the philosophers can be holding onto the fork. The rest of the specification describes what is the behavior of each action.

In [7] we have described a complete specification of the diarrheic philosophers. The specification of a philosopher that we provide below is slightly different to the one given earlier, in part since we abstract from some details to focus on the specification of the violations and their properties. Note that here we take a simpler approach: a philosopher can take both forks or none. The actions of the specification are the following.

$$\{\texttt{getthk}, \texttt{getbad}, \texttt{gethungry}, \texttt{up}_\texttt{L}, \texttt{up}_\texttt{R}\}$$

and they are intended to denote the same actions as [7]. We have the following propositions:

$$\{\mathtt{has_L}, \mathtt{has_R}, \mathtt{thk}, \mathtt{eating}, \mathtt{hungry}, \mathtt{bath}, \mathtt{has_L}, \mathtt{has_R}\}$$

and two violations $\{\mathtt{v}_1, \mathtt{v}_2\}$. The set of axioms of this specification is shown in figure 7. We consider the *GGG* predicate on the system axioms, i.e.:

Phil :

$\mathbf{p_1}$: $\mathsf{B} \to \neg\mathtt{v}_1 \wedge \neg\mathtt{v}_2 \wedge \mathtt{thk} \wedge \neg\mathtt{hungry} \wedge \neg\mathtt{bath}$

$\mathbf{p_2}$: $\mathtt{thk} \veebar \mathtt{hungry} \veebar \mathtt{eating} \veebar \mathtt{sick}$

$\mathbf{p_3}$: $\mathtt{eating} \leftrightarrow \mathtt{has_L} \wedge \mathtt{has_R} \wedge \neg\mathtt{sick}$

$\mathbf{p_4}$: $\neg\mathtt{hungry} \to \mathsf{AF}\mathtt{hungry}$

$\mathbf{p_5}$: $\neg\mathtt{eating} \to \mathsf{P}^1(\mathbf{U})$

$\mathbf{p_6}$: $\mathtt{eating} \to \mathsf{O}^1(\mathtt{down_L} \sqcap \mathtt{down_R})$

$\mathbf{p_7}$: $\mathsf{F}^1\mathtt{down_L} \to \boxed{\mathtt{down_L}}\mathtt{v}_1$

$\mathbf{p_8}$: $\mathsf{F}^1\mathtt{down_R} \to \boxed{\mathtt{down_R}}\mathtt{v}_2$

$\mathbf{p_9}$: $\mathtt{v}_1 \to \boxed{\mathtt{down_L}}\mathtt{v}_1$

$\mathbf{p_{10}}$: $\mathtt{v}_2 \to \boxed{\mathtt{down_R}}\mathtt{v}_2$

$\mathbf{p_{11}}$: $\mathtt{v}_1 \to [\mathtt{down_L}]\neg\mathtt{v}_1$

$\mathbf{p_{12}}$: $\mathtt{v}_2 \to [\mathtt{down_R}]\neg\mathtt{v}_2$

$\mathbf{p_{13}}$: $[\mathtt{getthk}]\mathtt{thk}$

$\mathbf{p_{15}}$: $\neg\mathtt{thk} \to [\mathtt{getthk}]\neg\mathtt{thk}$

$\mathbf{p_{16}}$: $[\mathtt{getbad}]\mathtt{bath}$

$\mathbf{p_{17}}$: $\neg\mathtt{bath} \to \boxed{\mathtt{getbad}}\neg\mathtt{bath}$

$\mathbf{p_{18}}$: $[\mathtt{gethungry}]\mathtt{hungry}$

$\mathbf{p_{19}}$: $\neg\mathtt{hungry} \to$
$[\mathtt{gethungry}]\neg\mathtt{hungry}$

$\mathbf{p_{20}}$: $\mathtt{thk} \to \mathtt{down_1} \wedge \mathtt{down_R}$

$\mathbf{p_{21}}$: $\mathtt{hungry} \to \mathtt{down_L} \wedge \mathtt{down_R}$

$\mathbf{p_{22}}$: $\mathtt{hungry} \wedge \langle\mathtt{up_L} \sqcup \mathtt{up_R}\rangle\top \to$
$\mathsf{AN}\mathtt{eating} \vee \mathsf{AN}\mathtt{hungry}$

$\mathbf{p_{23}}$: $\mathtt{bad} \to \boxed{\mathtt{getthk}}\mathtt{bad}$

$\mathbf{p_{24}}$: $\mathtt{eating} \to$
$\mathsf{AN}(\mathtt{thk} \vee \mathtt{bad})$

Figure 7: Specification of a philosopher

$\neg\mathtt{v}_i \wedge \mathsf{P}^1(\alpha) \to [\alpha]\neg\mathtt{v}_i$, for every i. Most of the axioms have already been introduced in the example given in [7]. We dicuss the remaining axioms. Axiom $\mathbf{P_4}$ says that a philosopher which 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 $\mathbf{p_7}$-$\mathbf{p_{12}}$ 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. In the same way that we did in [7], we simplify the problem by requiring that philosophers can only eat for a unit of time (axiom $\mathbf{p_{24}}$); the amount of time that the philosophers eat is not important for our current purposes. However, note that the specification can be extended establishing that a philosopher can only eat a finite amount of time (for this we need to have a component specifying a clock). In this case we will have a new violation in case the philosopher continues eating beyond the prescribed limit. (We can consider this a preferable violation to going to the bathroom!)

Now we focus on the analysis of violations. We suppose that some properties like $\vdash_{\mathbf{Phil}} \mathtt{v}_1 \to \mathtt{has_L}$ (if the philosopher is in violation $\mathtt{v}_1$, then he has the left fork) and $\vdash_{\mathbf{Phil}} \mathtt{v}_1 \to \neg\mathtt{eating}$ (if he is in a violation, then he is not eating) can be proven from the specification. (We proved these properties for the specification given in [7].)

Suppose that we want to obtain the specification of an 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 figure 8. The components $\mathbf{XFork^1}$ and $\mathbf{XFork^2}$

$$\begin{tikzcd}
\mathbf{XFork^1} & & \mathbf{Phil} & & \mathbf{XFork^2} \\
& \mathbf{Chan^1} \arrow[ul, "\tau_1"'] \arrow[ur, "\tau_2"] & & \mathbf{Chan^2} \arrow[ul, "\tau_1'"'] \arrow[ur, "\tau_2'"] &
\end{tikzcd}$$

Figure 8: Putting together forks with philosophers

are "instances" of the specification **XFork** (i.e., they obtained from **XFork** renaming the symbols with the subindex i), and $\mathbf{Chan^1}$ and $\mathbf{Chan^2}$ are "instances" of **Chan**. Here $\tau_1 : \mathbf{Chan} \to \mathbf{XFork}$ maps $\mathtt{port}^1_1 \mapsto \mathtt{lup}$ and $\mathtt{port}^1_2 \mapsto \mathtt{ldown}$, whereas $\tau_2 : \mathbf{Chan} \to \mathbf{Phil}$ maps $\mathtt{port}^1_1 \mapsto \mathtt{up}_\mathtt{L}$ and $\mathtt{port}^1_2 \mapsto \mathtt{down}_\mathtt{L}$ and similarly for τ_1' and τ_2'. In other words, these morphisms connect the right and the left fork with the philosopher. Let us call the colimit object of this specification **FPhil**, where the morphism $f_1 : \mathbf{XFork} \to \mathbf{FPhil}$, $f_2 : \mathbf{XFork} \to \mathbf{FPhil}$, $p_1 : \mathbf{Phil} \to \mathbf{FPhil}$, $c_1 : \mathbf{Chan} \to \mathbf{FPhil}$ and $c_2 : \mathbf{Chan} \to \mathbf{FPhil}$, are the required morphisms from the base of the cocone to the colimit object. Now, the upgrading diagram of **FPhil** is as shown in figure 9. The formulae at the right in this

$$\begin{tikzcd}
& \emptyset & \\
\mathtt{v}_1 \arrow[ur] & & \mathtt{v}_2 \arrow[ul] \\
& \mathtt{v}_1, \mathtt{v}_2 \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

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

Figure 9: Upgrading diagram of **FPhil**

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.

1.	$\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \to \neg\mathtt{eating}$	Property of **FPhil**
2.	$\vdash_{\mathbf{FPhil}} \neg\mathtt{eating} \to \mathsf{P}^1(\mathbf{U})$	$\mathbf{p_5}$
3.	$\vdash_{\mathbf{FPhil}} \mathsf{P}^1(\mathbf{U}) \to \mathsf{P}^1(\mathtt{down}_\mathtt{L})$	DPL
4.	$\vdash_{\mathbf{FPhil}} \neg\mathtt{v}_2 \wedge \mathsf{P}^1(\mathtt{down}_\mathtt{L}) \to [\mathtt{down}_\mathtt{L}]\neg\mathtt{v}_2$	GGG
5.	$\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \wedge \neg\mathtt{v}_2 \to [\mathtt{down}_\mathtt{L}]\neg\mathtt{v}_2$	ML, 1, 2, 4
6.	$\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \wedge \neg\mathtt{v}_2 \to [\mathtt{down}_\mathtt{L}]\neg\mathtt{v}_1 \wedge \neg\mathtt{v}_2$	PL, $\mathbf{p_{11}}$, 5
7.	$\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \to \mathtt{has}_L$	Property of **Phil**
8.	$\vdash_{\mathbf{XFork}} \mathtt{lup?} \to \langle\mathtt{ldown}\rangle\top$	$\mathbf{f_{14}}$
9.	$\vdash_{\mathbf{FPhil}} \mathtt{has}_\mathtt{L} \to \langle\mathtt{down}_\mathtt{L}\rangle\top$	Def.τ_{f_1} & theorem 11
10.	$\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \to \langle\mathtt{down}_\mathtt{L}\rangle\top$	PL, 7, 9
11.	$\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \wedge \neg\mathtt{v}_2 \to \langle\mathtt{down}_\mathtt{L}\rangle\top$	PL, 10
12.	$\vdash_{\mathbf{FPhil}} \mathtt{v}_1 \wedge \neg\mathtt{v}_2 \to [\mathtt{down}_\mathtt{L}](\neg\mathtt{v}_1 \wedge \neg\mathtt{v}_2) \wedge \langle\mathtt{down}_\mathtt{L}\rangle\top$	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 *GGG* property. 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 figure 10. The colimit of this diagram gives us the final design (say

$$\begin{tikzcd}
& \mathbf{XFork^1} & \\
\mathbf{Channel^1} \arrow[ur, "l_1"] \arrow[d, "o_1"'] & & \mathbf{Channel^3} \arrow[ul, "r_2"'] \arrow[d, "o_2"] \\
\mathbf{Phil^1} & & \mathbf{Phil^2} \\
\mathbf{Channel^2} \arrow[u, "o_3"] \arrow[dr, "r_1"'] & & \mathbf{Channel^4} \arrow[u, "o_4"'] \arrow[dl, "l_2"] \\
& \mathbf{XFork^2} &
\end{tikzcd}$$

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

Figure 10: Two philosophers eating

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 [12]). The interesting point here is to analyze 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 17 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 figure 11 This means that each of the (formulae which act

$$\begin{tikzcd}
& & & \emptyset & & & \\
\mathrm{v}_1^1 \arrow[urrr] & & \mathrm{v}_2^1 \arrow[ur] & & \mathrm{v}_1^2 \arrow[ul] & & v_2^2 \arrow[ulll] \\
& \mathrm{v}_1^1, \mathrm{v}_2^1 \arrow[ul] \arrow[ur] & & & & \mathrm{v}_1^2, \mathrm{v}_2^2 \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Figure 11: Coproduct of upgrading diagrams.

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 17 and 19 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 analyze the violation structure of a specification made from several components).

6 Further Remarks.

In this report, we have taken further the formalism presented in the earlier reports, using the ideas of [12] to allow the specification of different modules which can be put together to build a whole system. Also, we have provided a theoretical basis for analyzing the structure of the violations that can arise when executing an implementation of a specification. The main benefit of this formal machinery is that it allows us to deduce part of the violation structure of the system using the properties of the components.

It is important to stress that we have introduced a notion of bisimulation which allows us to characterize certain semantic structures which, in some way, reflect the locality of the components. Moreover, the deontic part of the original logic was generalized to allow a local version of permission and obligation, which is coherent with modularization, in the sense that the permissions and obligations of a given component do not affect (at least as far as this is desired) other parts of the system. In addition, having several versions of obligation and permission allows us to address several contrary-to-duty paradoxes, since by using different obligations we can avoid having contradictory deontic constructions. (The same idea of using different versions of deontic predicates to deal with contrary-to-duty paradoxes is proposed in [18].)

References

[1] Howard Barringer. The use of temporal logic in the compositional specification of concurrent systems. In A.Galton, editor, *Temporal Logic and their Applications*. Academic Press, 1987.

[2] P. BlackBurn, M.de Rijke, and Y.de Venema. *Modal Logic*. Cambridge Tracts in Theorical Computer Science 53, 2001.

[3] Michael C. Browne, Edmund M. Clarke, and Orna Grumberg. Characterizing kripke structures in temporal logic. *APSOFT'87: Proceedings of the International Joint Conference on Theory and Practice of Software Development, Pisa, Italy.*, 1987.

[4] R. Burstall and J. Goguen. Putting theories together to make specifications. In R.Reddy, editor, *Porc. Fifth International Joint Conference on Artificial Intelligence*, 1977.

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

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

[7] P.F. Castro and T.S.E. Maibaum. Towards a deontic logic for fault tolerance. Technical Report SQRL39, McMaster, Department of Computing & Software, Software Quality Research Laboratory, 2007.

[8] C. C. Chang and H. J. Keisler. *Model Theory.* North-Holland, 1973.

[9] Rocco DeNicola and Frits Vaandrager. Three logics for branching bisimulation. *Journal of the ACM*, 42:458–487, 1995.

[10] E.A. Emerson. *A Mathematical Introduction to Logic.* Academic Press, 1972.

[11] J. Fiadeiro and T.S.E. Maibaum. Towards object calculi. In Sernadas A Saake G, editor, *Information Systems; Correctness and Reusability.* Technische Universität Braunschweig, 1991.

[12] J. Fiadeiro and T.S.E. Maibaum. Temporal theories as modularization units for concurrent system specification. In *Formal Aspects of Computing*, volume 4, pages 239–272, 1992.

[13] J. Fiadeiro and Amílcar Sernadas. Structuring theories on consequence. In *Recent Trends in Data Type Specification, 5th Workshop on Abstract Data Types, Gullane, Scotland, Selected Papers*, pages 44–72, 1987.

[14] J.A. Goguen and R.M. Burstall. Institutions: Abstract model theory for specification and programming. In *Journal of the Association of Computing Machinery*, 1992.

[15] G. E. Hughes and M. J. Cresswell. *A New Introduction to Modal Logic.* Routledge, 1996.

[16] E. A. Emerson M. C. Browne and O. Grumberg. Characterizing finite kripke structures in propositional temporal logics. *Theoret. Comput. Sci.*, 59:115–131, 1988.

[17] Saunders MacLane and Ieke Moerdijk. *Sheaves in Geometry and Logic.* Springer-Verlag, 1992.

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

[19] R. Milner. *A Calculus of Communicating Systems.* Springer-Verlag, 1979.

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

[21] J. van Benthem. *Modal Correspondence Theory*. PhD thesis, Mathematisch Instituut & Instituut voor Gronslagenonderzoek, University of Amsterdam, 1976.

[22] Rob J. van Glabbeek and W. P. Weijland. Refinement in branching time semantics. In *Proceedings of the AMAST Conference. Iowa*, 1989.