Stepwise Development of Paraconsistent Processes

Juliana Cunha[1(✉)], Alexandre Madeira[1(✉)], and Luís Soares Barbosa[2(✉)]

[1] CIDMA, Department of Mathematics, Aveiro University, Aveiro, Portugal
juliana.cunha@live.ua.pt, madeira@ua.pt
[2] INESC TEC & Department of Informatics, Minho University, Braga, Portugal
lsb@di.uminho.pt

Abstract. The development of more flexible and robust models for reasoning about systems in environments with potentially conflicting information is becoming more and more relevant in different contexts. In this direction, we recently introduced paraconsistent transition systems, i.e. transition systems whose transitions are tagged with a pair of weights, one standing for the degree of evidence that the transition exists, another weighting its potential non existence. Moreover, these structures were endowed with a modal logic [3] that was further formalised as an institution in [5]. This paper goes a step further, proposing an approach for the structured specification of paraconsistent transition processes, i.e. paraconsistent transition systems with initial states. The proposed approach is developed along the lines of [12], which introduced a complete methodology for (standard) reactive systems development building on the Sannella and Tarlecki stepwise implementation process. For this, we enrich the logic with dynamic modalities and hybrid features, and provide a pallet of constructors and abstractors to support the development process of paraconsistent processes along the entire design cycle.

1 Introduction

The development of more flexible and robust models for reasoning about systems in environments with potentially conflicting information is becoming more and more relevant in different contexts. Applications scenarios where these patterns emerge range from robotics (e.g. specifying a controller that has to react to collected conflicting data due faulty or imprecise sensors), to diagnostic support systems for the health domain (e.g. often dealing with contradictory marks). Moreover, it is not uncommon that along a development process, engineering

The present study was developed in the scope of the Project Agenda ILLIANCE [C644919832-00000035 — Project n 46], financed by PRR - Plano de Recuperação e Resiliência under the Next Generation EU from the European Union.
FCT, the Portuguese funding agency for Science and Technology suports the second author with the project UIDB/04106/2020 and the third with the project IBEX PTDC/CCI-COM/4280/2021.

C. David and M. Sun (Eds.): TASE 2023, LNCS 13931, pp. 327–343, 2023.
https://doi.org/10.1007/978-3-031-35257-7_20

teams make design decisions on often imprecise and contradictory requirements, elicited from stakeholders with distinct perspectives.

This entails the need for the design of new models and logics resilient to conflict, as well as methods prepared to assist the rigorous development and analysis of such complex systems. Following this direction, we introduced in [4] the notion of a paraconsistent transition system, where transitions are labelled with a pair of weights, one weighting the degree of evidence that the transition exists, another weighting its potential non existence. Then, in [3] these structures were endowed with a modal logic, that was further extended to a multi-modal logic, and formalised as an institution, in [5]. The latter reference, also discusses preliminary steps towards the structured specification of these structures, by characterizing a set of CASL-like operators.

The present paper contributes to this agenda along two main edges. Firstly, we adjust the models and extend the logic to express and reason about paraconsistent processes, i.e. processes with paraconsistent weighted transitions with specified initial states. Moreover, the multi-modal logic described in [5] is extended to a dynamic logic [10]. This involves bringing into the picture structured modalities, to express abstract properties such as safety and liveness, as well as hybrid logic constructs [2], such as binders and state-variables to express some other properties of paraconsistent processes as concrete transitions within concrete states. In the end, we end up with a paraconsistent version of the logic introduced in [12], which we also formalise as a logical institution.

Secondly, we introduce a method for the structured specification of paraconsistent transition processes starting from their abstract design down to the concrete implementation stage. More precisely, we fully instantiate the (generic) Sannella and Tarlecki's stepwise implementation process [15] for the case of development of paraconsistent processes. On the base of the approach of Sannella and Tarlecki is the notion of a specification as a syntactic representation of a class of possible implementations of the envisaged system. Conceptually, the implementation process corresponds to a chain of refinement steps in the sense that a specification refines another one if every model of the latter is a model of the former. This is complemented by the concepts of *constructor* and *abstractor* implementations, conveying the idea that a specification may resort to one or several given specifications applying a specific construction on top of them to meet the envisaged requirements. Thus, this paper also considers abstractor implementations to capture situations whose relevant requirements just need to be satisfied up to some suitable abstraction (e.g. freed from some implementation details). Note that this is not a trivial step since even the notion of a specification characterized by a class of models has to be adjusted to the proposed paraconsistent framework.

Finally, a pallet of constructors and abstracts was defined to support the whole development process. We revisited the pallet of process constructors and abstractors discussed in [12] for the development of (standard) processes, redefining their semantics in the paraconsistent setting. This includes operators to relabel and hidden actions, but also to compose specification using the product

constructor. Bisimulation of paraconsistent transition structures introduced in [4] is used to support abstractor implementations.

The rest of the paper is organised as follows: Sect. 2 introduces some background definitions for the development of the work. Then, Sect. 3 introduces $\boldsymbol{P}(\mathcal{A})$, an institution suitable to express and reason about paraconsistent transition systems. Section 4 depicts a complete formal development *à la* Sannella and Tarlecki for these systems. Finally, Sect. 5 concludes.

2 Preliminaries

Let us start by recalling what an institution is. Then, we will introduce a specific algebraic structure over which the logical system to support our specification will be defined. Such a structure is basically a particular class of residuated lattices in which the lattice meet and the monoidal composition coincide, equipped with a metric which measures the "distance" between specification which may be fully consistent (in which case the weights in the double transitions considered are complementary), vague (when their "sum" remains below the entire universe of discourse, typically sum less than 1), or paraconsistent (when sums above).

Institutions. Informally, an institution abstractly defines a logic system by describing through its spectrum of signatures, corresponding models and a satisfaction relation between models and sentences.

Definition 1 ([9])**.** *An institution I is a tuple $I = (\mathsf{Sign}_I, \mathsf{Sen}_I, \mathsf{Mod}_I, \models_I)$ consisting of*

- *a category Sign_I of signatures*
- *a functor $\mathsf{Sen}_I : \mathsf{Sign}_I \to \mathbb{S}et$ giving a set of Σ-sentences for each signature $\Sigma \in |\mathsf{Sign}_I|$. For each signature morphism $\sigma : \Sigma \to \Sigma'$ the function $\mathsf{Sen}_I(\sigma) : \mathsf{Sen}_I(\Sigma) \to \mathsf{Sen}_I(\Sigma')$ translates Σ-sentences to Σ'-sentences*
- *a functor $\mathsf{Mod}_I : \mathsf{Sign}_I^{op} \to Cat$ assigns to each signature Σ the category of Σ-models. For each signature morphism $\sigma : \Sigma \to \Sigma'$ the functor $\mathsf{Mod}_I(\sigma) : \mathsf{Mod}_I(\Sigma') \to \mathsf{Mod}_I(\Sigma)$ translates Σ'-models to Σ-models.*
- *a satisfaction relation $\models_I^{\Sigma} \subseteq |\mathsf{Mod}_I(\Sigma)| \times \mathsf{Sen}_I(\Sigma)$ determines the satisfaction of Σ-sentences by Σ-models for each signature $\Sigma \in |\mathsf{Sign}_I|$.*

such that satisfaction is preserved under change of signature, that is for any signature morphism $\sigma : \Sigma \to \Sigma'$, for any $\varphi \in \mathsf{Sen}_I(\Sigma)$ and $M' \in |\mathsf{Mod}_I(\Sigma')|$

$$\left(M' \models_I^{\Sigma'} \mathsf{Sen}_I(\sigma)(\varphi)\right) \Leftrightarrow \left(\mathsf{Mod}_I(\sigma)(M') \models_I^{\Sigma} \varphi\right) \tag{1}$$

When formalising multi-valued logics as institutions, the equivalence on the satisfaction condition (1) can be replaced by an equality [1]:

$$\left(M' \models_I^{\Sigma'} \mathsf{Sen}_I(\sigma)(\varphi)\right) = \left(\mathsf{Mod}_I(\sigma)(M') \models_I^{\Sigma} \varphi\right) \tag{2}$$

(Metric) Twisted Algebras. Metric twisted algebras are introduced to deal with weights in paraconsistent transition systems.
A residuated lattice $\langle A, \sqcap, \sqcup, 1, 0, \odot, \rightharpoonup, e\rangle$, over a nonempty set A, is a complete lattice $\langle A, \sqcap, \sqcup\rangle$, equipped with a monoid $\langle A, \odot, e\rangle$ such that $\odot$ has a right adjoint, $\rightharpoonup$, called the residuum. We will focus on a class of complete residuated lattices that are bounded by a maximal element 1 and a minimal element 0 and that are *integral*, that is $1 = e$. Additionally, we want the lattice meet ($\sqcap$) and the monoidal composition ($\odot$) to coincide. Hence, the adjunction is stated as $a \sqcap b \leqslant c$ iff $b \leqslant a \rightharpoonup c$. A pre-linearity condition is also enforced

$$(a \rightharpoonup b) \sqcup (b \rightharpoonup a) = 1 \tag{3}$$

A residuated lattice obeying prelinearity is known as a MTL-algebra [7], with a slight abuse of nomenclature, the designation iMTL-algebra, from *integral MTL-algebra*, will be used in the sequel for the class of semantic structures considered. Examples of iMTL-algebras are:

- the Boolean algebra $\mathbf{2} = \langle \{0, 1\}, \wedge, \vee, 1, 0, \rightarrow\rangle$
- $\mathbf{3} = \langle \{\top, u, \bot\}, \wedge_3, \vee_3, \top, \bot, \rightarrow_3\rangle$, where

$\wedge_3$	$\bot$	u	$\top$
$\bot$	$\bot$	$\bot$	$\bot$
u	$\bot$	u	u
$\top$	$\bot$	u	$\top$

$\vee_3$	$\bot$	u	$\top$
$\bot$	$\bot$	u	$\top$
u	u	u	$\top$
$\top$	$\top$	$\top$	$\top$

$\rightarrow_3$	$\bot$	u	$\top$
$\bot$	$\top$	$\top$	$\top$
u	$\bot$	$\top$	$\top$
$\top$	$\bot$	u	$\top$

- $\ddot{\boldsymbol{G}} = \langle [0, 1], \min, \max, 1, 0, \rightarrow\rangle$, with implication defined as

$$a \rightarrow b = \begin{cases} 1 & if\, a \leqslant b \\ b & otherwise \end{cases}$$

We focus on iMTL-algebras $\mathbf{A}$ whose carrier A supports a metric space (A, d), with suitable choice of d. Where $d\colon A \times A \to \mathbb{R}^+$ such that $d(x, y) = 0$ iff $x = y$ and $d(x, y) \leqslant d(x, z) + d(z, y)$. The notion of a $\mathbf{A}$-twisted algebra, was introduced in [3] to operate with pairs of truth weights, which consists of an enrichment of a twist-structure [11] with a metric. This metric is necessary to the characterization of the consistency of these pairs of values, relevant on the interpretation of the consistency operator $\circ$ in the logic (see [3] for details).

Definition 2 ([3]). *Given a iMTL-algebra $\boldsymbol{A}$ enriched with a metric d, a $\boldsymbol{A}$-twisted algebra $\mathcal{A} = \langle A \times A, \boxplus, \boxplus, \Rightarrow, /\!/, D\rangle$ is defined as: $(a, b) \boxplus (c, d) = (a \sqcap c, b \sqcup d)$, $(a, b) \boxplus (c, d) = (a \sqcup c, b \sqcap d)$, $(a, b) \Rightarrow (c, d) = (a \rightharpoonup c, a \sqcap d)$, $/\!/(a, b) = (b, a)$ and $D((a, b), (c, d)) = \sqrt{d(a, c)^2 + d(b, d)^2}$. The order in $\boldsymbol{A}$ is lifted to $\mathcal{A}$ as $(a, b) \preceq (c, d)$ iff $a \leqslant c$ and $b \geqslant d$.*

3 An Institution for Paraconsistent Transitions Processes

This section introduces $\boldsymbol{P}(\mathcal{A})$, a logic for paraconsistent processes formalized as a (many-valued) institution. $\boldsymbol{P}(\mathcal{A})$ starting point is the logic for paraconsistent systems presented in [5]. However, in $\boldsymbol{P}(\mathcal{A})$ modalities can be indexed by regular

expressions of actions, as in dynamic logic, and a binder $\downarrow x$ and identification $@_x$ operator, borrowed from hybrid logic, are introduced.

Let us fix any twisted algebra $\mathcal{A}$ to introduce all the necessary ingredients for an institution $\boldsymbol{P}(\mathcal{A}) = (\mathsf{Sign}, \mathsf{Sen}, \mathsf{Mod}, \models)$. Whenever the choice of $\mathcal{A}$ is not essential, $\boldsymbol{P}(\mathcal{A})$ will be abbreviated to $\boldsymbol{P}$. Note here that all constructions of $\boldsymbol{P}(\mathcal{A})$ are parametrically defined, thus admitting different instances according to the structure of the truth values domain relevant for the application at hands.

Firstly we introduce the signatures:

Definition 3. *A signature is a tuple* $\langle \mathrm{Act}, \mathrm{Prop} \rangle$ *where* Act *is a finite set of action symbols and* Prop *is a set of propositions.*

The set of actions $\mathrm{Act} = \{a_1, \ldots, a_n\}$ will induce a set of structured actions, $Str(\mathrm{Act})$, defined by $\alpha := a \mid \alpha; \alpha \mid \alpha + \alpha \mid \alpha^*$ where $a \in Act$. As usual, we use $-a_i$ to denote the structured action $a_1 + \cdots + a_{i-1} + a_{i+1} + \cdots + a_n$ and, given a set of atomic actions $B = \{b_1, \ldots b_k\} \subseteq \mathrm{Act}$, we write B to refer to the structured action $b_1 + \cdots + b_k$.

Let $\Sigma, \Sigma' \in \mathsf{Sign}$ be signatures. A signature morphism $\sigma : \Sigma \to \Sigma'$ involves two functions $\sigma_{\mathrm{Prop}} : \mathrm{Prop} \to \mathrm{Prop}'$ and $\sigma_{\mathrm{Act}} : \mathrm{Act} \to \mathrm{Act}'$, where σ_{Act} extends to $Str(\mathrm{Act})$ as follows:

- $\widehat{\sigma}_{\mathrm{Act}}(a) = \sigma_{\mathrm{Act}}(a)$
- $\widehat{\sigma}_{\mathrm{Act}}(\alpha; \alpha') = \widehat{\sigma}_{\mathrm{Act}}(\alpha); \widehat{\sigma}_{\mathrm{Act}}(\alpha')$
- $\widehat{\sigma}_{\mathrm{Act}}(\alpha + \alpha') = \widehat{\sigma}_{\mathrm{Act}}(\alpha) + \widehat{\sigma}_{\mathrm{Act}}(\alpha')$
- $\widehat{\sigma}_{\mathrm{Act}}(\alpha^*) = \widehat{\sigma}_{\mathrm{Act}}(\alpha)^*$

$$\begin{array}{ccc} \mathrm{Act} & \xrightarrow{\sigma_{\mathrm{Act}}} & \mathrm{Act}' \\ \downarrow & & \downarrow \\ Str(\mathrm{Act}) & \xrightarrow{\widehat{\sigma}_{\mathrm{Act}}} & Str(\mathrm{Act}') \end{array}$$

for all $a \in \mathrm{Act}$ and $\alpha, \alpha' \in Str(\mathrm{Act})$. The category of signatures and their morphisms form category Sign.

Now, let us introduce the models:

Definition 4. *A* $\langle \mathrm{Act}, \mathrm{Prop} \rangle$*-paraconsistent transition process, abbreviated to PTP, is a tuple* $P = (W, w_0, R, V)$ *such that,*

- W *is a non-empty set of states,*
- $w_0 \in W$ *is called the initial state*
- $R = (R_a \colon W \times W \to A \times A)_{a \in \mathrm{Act}}$ *is an* Act*-indexed family of functions;* $R_a(w_1, w_2) = (\mathit{tt}, \mathit{ff})$ *means that there is the evidence degree* tt *that there exists a transition from* w_1 *to* w_2 *by* a*, and the evidence degree* ff *that this transition does not exists.*
- $V : W \times \mathrm{Prop} \to A \times A$ *is a valuation function;* $V(w, p) = (\mathit{tt}, \mathit{ff})$ *means that in state* w *there is the evidence degree* tt *that the proposition* p *holds, and* ff *that it does not hold.*

For any pair $(\mathit{tt}, \mathit{ff}) \in A \times A$*,* $(\mathit{tt}, \mathit{ff})^+$ *denotes* tt *and* $(\mathit{tt}, \mathit{ff})^-$ *denotes* ff*.*

The interpretation of $\alpha \in Str(\mathrm{Act})$ in a model (W, w_0, R, V) extends the relation R to a relation $\widehat{R}$ defined for states $w, w' \in W$ as:

- $\widehat{R}_a(w,w') = R_a(w,w')$, for $a \in \mathrm{Act}$
- $\widehat{R}_{\alpha+\alpha'}(w,w') = \widehat{R}_\alpha(w,w') \amalg \widehat{R}_{\alpha'}(w,w')$
- $\widehat{R}_{\alpha;\alpha'}(w,w') = \coprod_{v \in W} \left(R_\alpha(w,v) \pitchfork R_{\alpha'}(v,w') \right)$
- $\widehat{R}_{\alpha*}(w,w') = \coprod_{i \geqslant 0} \widehat{R}^i_\alpha(w,w')$. Where,
  - $\widehat{R}^0_\alpha = \begin{cases} (1,0) & \text{if } w = w' \\ (0,1) & \text{otherwise} \end{cases}$
  - $\widehat{R}^{k+1}_\alpha(w,w') = (\widehat{R}^k_\alpha ; \widehat{R}_\alpha)(w,w')$

where $\overline{\pitchfork}$ and $\overline{\amalg}$ are the distributed versions of $\pitchfork$ and $\amalg$, respectively.

Definition 5. *A morphism connecting two $\langle \mathrm{Act}, \mathrm{Prop} \rangle$-PTPs (W, w_0, R, V) and (W', w'_0, R', V') is a function $h : W \to W'$, such that: for each $a \in \mathrm{Act}$, $R_a(w_1, w_2) \preccurlyeq R'_a(h(w_1), h(w_2))$; for any $p \in \mathrm{Prop}$, $w \in W$, $V(w,p) \preccurlyeq V'(h(w), p)$; and $h(w_0) = w'_0$.*

We say that P and P' are isomorphic, in symbols $P \cong P'$, whenever there are morphisms $h : W \to W'$ and $h^{-1} : W' \to W$ such that $h' \circ h = id_{W'}$ $h \circ h' = id_W$.

Models and their corresponding morphisms form a category, denoted by Mod, which acts as the model category for our institution $\boldsymbol{P}$.

Definition 6. *For any signature morphism $\sigma : \langle \mathrm{Act}, \mathrm{Prop} \rangle \to \langle \mathrm{Act}', \mathrm{Prop}' \rangle$ and $P = (W', w'_0, R', V')$ a $\langle \mathrm{Act}', \mathrm{Prop}' \rangle$-PTP, the σ-reduct of P' is a $\langle \mathrm{Act}, \mathrm{Prop} \rangle$-PTP $P|_\sigma = (W, w_0, R, V)$ such that $W = W'$, $w_0 = w'_0$; for $p \in \mathrm{Prop}$, $w \in W$, $V(w,p) = V'(w, \sigma(p))$; and for $w, v \in W$ and $a \in \mathrm{Act}$, $R_a(w,v) = R'_{\sigma(a)}(w,v)$.*

Therefore, each signature morphism $\sigma : \Sigma \to \Sigma'$ defines a functor $\mathsf{Mod}(\sigma) : \mathsf{Mod}(\Sigma') \to \mathsf{Mod}(\Sigma)$ that maps processes and morphisms to the corresponding reducts. This lifts to a functor, $\mathsf{Mod} : (\mathsf{Sign})^{op} \to \mathsf{CAT}$, mapping each signature to the category of its models, and each signature morphism to its reduct functor.

Definition 7. *Given a signature $\Sigma = \langle \mathrm{Act}, \mathrm{Prop} \rangle$ the set $\mathsf{Sen}(\Sigma)$ of sentences is given by the following grammar*

$$\varphi ::= p \mid \bot \mid \neg\varphi \mid \varphi \to \varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid [\alpha]\,\varphi \mid \langle\alpha\rangle\varphi \mid \cancel{[\alpha]}\,\varphi \mid \cancel{\langle\alpha\rangle}\varphi \mid \circ\varphi \mid x \mid \downarrow x.\varphi \mid @_x\varphi$$

where $p \in Prop$, $\alpha \in Str(\mathrm{Act})$ and $x \in X$, with X being an infinite set of variables.

Each signature morphism $\sigma : \langle \mathrm{Act}, \mathrm{Prop} \rangle \to \langle \mathrm{Act}', \mathrm{Prop}' \rangle$ induces a sentence translation scheme $\mathsf{Sen}(\sigma)$ recursively defined as follows:

- $\mathsf{Sen}(\sigma)(p) = \sigma_{\mathrm{Prop}}(p)$
- $\mathsf{Sen}(\sigma)(\bot) = \bot$
- $\mathsf{Sen}(\sigma)(\neg\varphi) = \neg\mathsf{Sen}(\sigma)(\varphi)$
- $\mathsf{Sen}(\sigma)(\varphi \odot \varphi') = \mathsf{Sen}(\sigma)(\varphi) \odot \mathsf{Sen}(\sigma)(\varphi')$, $\odot \in \{\vee, \wedge, \to\}$
- $\mathsf{Sen}(\sigma)([\alpha]\varphi) = [\widehat{\sigma}_{\mathrm{Act}}(\alpha)]\ \mathsf{Sen}(\sigma)(\varphi)$

- $\mathsf{Sen}(\sigma)(\langle\alpha\rangle\varphi) = \langle\widehat{\sigma}_{\mathrm{Act}}(\alpha)\rangle\,\mathsf{Sen}(\sigma)(\varphi)$
- $\mathsf{Sen}(\sigma)([\not{\alpha}]\varphi) = [\widehat{\sigma}_{\mathrm{Act}}(\alpha)]\,\mathsf{Sen}(\sigma)(\varphi)$
- $\mathsf{Sen}(\sigma)(\langle\not{\alpha}\rangle\varphi) = \langle\widehat{\sigma}_{\mathrm{Act}}(\alpha)\rangle\,\mathsf{Sen}(\sigma)(\varphi)$
- $\mathsf{Sen}(\sigma)(\circ\varphi) = \circ\,\mathsf{Sen}(\sigma)(\varphi)$
- $\mathsf{Sen}(\sigma)(x) = x$
- $\mathsf{Sen}(\sigma)(\downarrow x.\varphi) = \downarrow x.\mathsf{Sen}(\sigma)(\varphi)$
- $\mathsf{Sen}(\sigma)(@_x\varphi) = @_x\mathsf{Sen}(\sigma)(\varphi)$

Entailing a functor $\mathsf{Sen} : \mathsf{Sign} \to \mathsf{Set}$ mapping each signature to the set of its sentences, and each signature morphism to the corresponding translation of sentences.

Definition 8. *Let $\Sigma = \langle \mathrm{Act}, \mathrm{Prop}\rangle$ be a signature, $P = (W, w_0, R, V)$ a Σ-PTP and φ a Σ-sentence, the satisfaction relation*

$$(P \models \varphi) \;=\; \prod_{g \in W^X} (P, g, w_0 \models \varphi)$$

where $g : X \to W$ is a valuation function such that, for $x \in X$, $g[x \mapsto w]$ denotes the valuation given by $g[x \mapsto w](x) = w$ and $g[x \mapsto w](y) = g(y)$ for any $y \neq x \in X$. If φ is a formula without free variables then the valuation function g is irrelevant, that is, $(M, g, w \models \varphi) = (M, w \models \varphi)$. The relation $\models$ is recursively defined as follows

- $(M, w \models p) = V(w, p)$
- $(M, w \models \bot) = (0, 1)$
- $(M, w \models \neg\varphi) = /\!\!/(M, w \models \varphi)$
- $(M, w \models \varphi \to \varphi') = (M, w \models \varphi) \Rightarrow (M, w \models \varphi')$
- $(M, w \models \varphi \vee \varphi') = (M, w \models \varphi) \sqcup (M, w \models \varphi')$
- $(M, w \models \varphi \wedge \varphi') = (M, w \models \varphi) \sqcap (M, w \models \varphi')$
- $(M, w \models \circ\varphi) = \begin{cases} (1,0) & if\ (M, w \models \varphi) \in \Delta_C \\ (0,1) & otherwise \end{cases}$
- $(M, g, w \models [\alpha]\varphi) = (\,[\alpha^+](M, g, w, \varphi^+), \langle\alpha^+\rangle(M, g, w, \varphi^-)\,)$
- $(M, g, w \models \langle\alpha\rangle\varphi) = (\,\langle\alpha^+\rangle(M, g, w, \varphi^+), [\alpha^+](M, g, w, \varphi^-)\,)$
- $(M, g, w \models [\not{\alpha}]\varphi) = (\,\langle\alpha^-\rangle(M, g, w, \varphi^-), [\alpha^-](M, g, w, \varphi^+)\,)$
- $(M, g, w \models \langle\not{\alpha}\rangle\varphi) = (\,[\alpha^-](M, g, w, \varphi^-), \langle\alpha^-\rangle(M, g, w, \varphi^+)\,)$
- $(M, g, w \models x) = (1, 0)$ *iff* $g(x) = w$
- $(M, g, w \models \downarrow x.\varphi) = (M, g[x \mapsto w], w \models \varphi)$
- $(M, g, w \models @_x\varphi) = (M, g, g(x) \models \varphi)$

where

- $[\alpha^+](M, g, w, \varphi^*) = \prod_{w' \in W} (\widehat{R}^+_\alpha(w, w') \rightharpoonup (M, g, w' \models \varphi)^*)$
- $[\alpha^-](M, g, w, \varphi^*) = \prod_{w' \in W} (\widehat{R}^-_\alpha(w, w') \rightharpoonup (M, g, w' \models \varphi)^*)$
- $\langle\alpha^+\rangle(M, g, w, \varphi^*) = \bigsqcup_{w' \in W} (\widehat{R}^+_\alpha(w, w') \sqcap (M, g, w' \models \varphi)^*)$
- $\langle\alpha^-\rangle(M, g, w, \varphi^*) = \bigsqcup_{w' \in W} (\widehat{R}^-_\alpha(w, w') \sqcap (M, g, w' \models \varphi)^*)$

– $\Delta_C = \{(a,b) \mid D((a,b),(0,0)) \leqslant D((a,b),(1,1))\}$

with $* \in \{^+, ^-\}$, $\alpha \in Str(\mathrm{Act})$ *and* $\bigsqcup$ *and* $\bigsqcap$ *are the distributed versions of* $\sqcup$ *and* $\sqcap$, *respectively.*

Proposition 1. *Let* $\Sigma = \langle \mathrm{Act}, \mathrm{Prop} \rangle$ *and* $\Sigma' = \langle \mathrm{Act}', \mathrm{Prop}' \rangle$ *be signatures and* $\sigma : \Sigma \to \Sigma'$ *be a signature morphism. For any* $P' = (W', w_0', R', V') \in \mathsf{Mod}(\Sigma')$ *and* $\varphi \in \mathsf{Sen}(\Sigma)$,

$$\big(P'|_\sigma \models \varphi\big) \;=\; \big(P' \models \mathsf{Sen}(\sigma)(\varphi)\big) \tag{4}$$

Proof. For any $w \in W$ $\big(P'|_\sigma, g, w \models \varphi\big) \;=\; \big(P', g, w \models \mathsf{Sen}(\sigma)(\varphi)\big)$. The proof of this statement is done by induction over the structure of sentences. Cases $\bot$ is trivial $\big(P'|_\sigma, g, w \models \bot\big) \;=\; \big(P', g, w \models \mathsf{Sen}(\sigma)(\bot)\big) = (0,1)$ and for any $p \in \mathrm{Prop}$, by the defn of σ-reduct, $V(w,p) = V'(w, \sigma_{\mathrm{Prop}}(p))$ which is equal to $\big(P'|_\sigma, g, w \models p\big) \;=\; \big(P', g, w \models \mathsf{Sen}(\sigma)(p)\big)$. For Boolean connectives $\neg$, $\wedge$, $\vee$, $\to$ and $\circ$ the proof boils down to first using the defn of Sen, followed by the defn of $\models$, the induction hypothesis and finally the defn of $\models$ again. For more details regarding this part of the proof we refer to [5] where a similar proof is done. For $[\alpha]\,\varphi$ the proof is similiar to the other modal connectives.

$$\begin{aligned}
&P', g, w \models \mathsf{Sen}(\sigma)([\alpha]\,\varphi)\\
=\;&\{\text{defn of } \mathsf{Sen}\}\\
&P', g, w \models [\widehat{\sigma}(\alpha)]\,\mathsf{Sen}(\sigma)(\varphi)\\
=\;&\{\text{defn of } \models\}\\
&([\widehat{\sigma}(\alpha)^+](P', g, w, \mathsf{Sen}(\sigma)(\varphi)^+)\,, \langle\widehat{\sigma}(\alpha)^+\rangle(P', g, w, \mathsf{Sen}(\sigma)(\varphi)^-))\\
=\;&\{\text{defn of } [\alpha^+] \text{ and } \langle\alpha^+\rangle\}\\
&\Big(\bigsqcap_{w' \in W'} (R'^+_{\widehat{\sigma}(a)}(w,w') \rightharpoonup (P', g, w' \models \mathsf{Sen}(\sigma)(\varphi))^+),\\
&\bigsqcup_{w' \in W'} (R'^+_{\widehat{\sigma}(a)}(w,w') \sqcap (P', g, w' \models \mathsf{Sen}(\sigma)(\varphi))^-)\Big)\\
=\;&\{(\text{step } \star)\}\\
&\Big(\bigsqcap_{w' \in W} (R^+_\alpha(w,w') \rightharpoonup (P'|_\sigma, g, w \models \varphi)^+),\ \bigsqcup_{w' \in W} (R^+_\alpha(w,w') \sqcap (P'|_\sigma, g, w \models \varphi)^-)\Big)\\
=\;&\{\text{defn } [\alpha^+] \text{ and } \langle\alpha^+\rangle\}\\
&([\alpha^+](P'|_\sigma, g, w, \varphi^+)\,, \langle\alpha^+\rangle(P'|_\sigma, g, w, \varphi^-))\\
=\;&\{\text{defn of } \models\}\\
&P'|_\sigma, g, w \models [\alpha]\,\varphi
\end{aligned}$$

(step $\star$) By the definition of reduct we have that $W = W'$ and for any w, $w' \in W$, $R'_{\widehat{\sigma}(\alpha)}(w,w') = R_\alpha(w,w') = (tt, ff)$. Also by the induction hypothesis, $(P', g, w \models \mathsf{Sen}(\sigma)(\varphi)) = (P'|_\sigma, g, w \models \varphi)$, which is equivalent to writing:

$$\begin{aligned}
\Big((P', g, w \models \mathsf{Sen}(\sigma)(\varphi))^+, (P', g, w \models \mathsf{Sen}(\sigma)(\varphi))^-)\Big) =\\
= \Big((P'|_\sigma, g, w \models \varphi)^+, (P'|_\sigma, g, w \models \varphi)^-\Big)
\end{aligned} \tag{5}$$

Therefore $P', g, w \models \mathsf{Sen}(\sigma)(\varphi))^+ = (P'|_\sigma, g, w \models \varphi)^+$ and $(P', g, w \models \mathsf{Sen}(\sigma)(\varphi))^- = (P'|_\sigma, g, w \models \varphi)^-$.
For the case of state variables we have that $P', g, w \models \mathsf{Sen}(\sigma)(x)$ is either $(1, 0)$ or $(0, 1)$. We illustrate the proof is it is equal to $(1, 0)$ then, if $(P', g, w \models \mathsf{Sen}(\sigma)(x)) = (1, 0)$, by definition of Sen, we have $(P', g, w \models x) = (1, 0)$. Thus, by induction hypothesis, $(P'|_\sigma, g, w \models x) = (1, 0)$.

The case $M', g, w \models \mathsf{Sen}(\sigma)(\downarrow x.\varphi)$ using the defn of Sen and $\models$, $M', g[x \mapsto w], w \models \mathsf{Sen}(\sigma)(\varphi)$ by the induction hypothesis and defn of $\models$, $M'|_\sigma, g, w \models \downarrow x.\varphi$. Similarly, for $M', g, w \models \mathsf{Sen}(\sigma)(@_x.\varphi) = M', g, w \models @_x.\mathsf{Sen}(\sigma)(\varphi)$ using the defn of $\models$, $M', g, g(x) \models \mathsf{Sen}(\sigma)(\varphi)$ followed by using the induction hypothesis and defn of $\models$, $M'|_\sigma, g, w \models @_x\varphi$.

In conclusion we have proven for any $w \in W$, $\big(P'|_\sigma, g, w \models \varphi\big) = \big(P', g, w \models \mathsf{Sen}(\sigma)(\varphi)\big)$ if w is replaced by w_0, $\big(M'|_\sigma \models \varphi\big) = \big(M' \models \mathsf{Sen}(\sigma)(\varphi)\big)$.

The following theorem is a consequence of this subsection where all the ingredients of institution $\boldsymbol{P}$ are formalized in a categorical manner.

Theorem 1. *$\boldsymbol{P}(\mathcal{A}) = (\mathsf{Sign}_{\boldsymbol{P}(\mathcal{A})}, \mathsf{Sen}_{\boldsymbol{P}(\mathcal{A})}, \mathsf{Mod}_{\boldsymbol{P}(\mathcal{A})}, \models_{\boldsymbol{P}(\mathcal{A})})$ is an institution, for any fixed twisted algebra $\mathcal{A}$.*

4 Formal Development Method *à la* Sannella and Tarlecki

Once set a suitable institution to reason about specifications of paraconsistent processes, we turn to the methodological level. Therefore, the notions of a *simple*, *constructor* and *abstract* implementation for paraconsistent specifications are presented below.

A paraconsistent specification consists of a pair $SP = (Sig(SP), Mod(SP))$ where $Sig(SP)$ is a signature in Sign and $Mod(SP) : \mathsf{Mod}(Sig(SP)) \to A \times A$ is a mapping that associates to any process $P \in \mathsf{Mod}(Sig(SP))$ a pair $(\mathit{tt}, \mathit{ff}) \in A \times A$ such that tt represents the evidence degree of P satisfying the requirements of SP and ff represents the evidence degree of P not satisfying the requirements of SP.

Definition 9. *A flat specification is a pair $SP = (\Sigma, \Phi)$ where $\Sigma \in \mathsf{Sign}$ is a signature and $\Phi \subseteq \mathsf{Sen}(\Sigma)$ is a set of axioms. Hence, $Sig(SP) = \Sigma$ and $Mod(SP)(P) = \left(\prod\!\!\!\prod_{\varphi \in \Phi} (P \models \varphi) \right)$.*

Flat specifications, that consist of a signature and a set of axioms, are suitable to capture requirements that can be easily expressed by a set of axioms. A simple refinement of specifications is classically seen as a restriction of the class of models. For paraconsistent specifications refinement is defined as follows

Definition 10. *A paraconsistent specification SP' is said to simple refine, or implement, another paraconsistent specification SP, in symbols $SP \rightsquigarrow SP'$, if both are over the same signature and for all $P \in \mathsf{Mod}(Sig(SP))$, $Mod(SP')(P) \preccurlyeq Mod(SP)(P)$.*

Transitivity of $\preccurlyeq$ ensures that vertical composition of simple implementations is well defined, that is, if $SP \rightsquigarrow SP'$ and $SP' \rightsquigarrow SP''$ then $SP \rightsquigarrow SP''$.

The running example in this work is adapted from the examples documented in [12] of a file compressing service. It consists of a compressing service of text files whose information regarding inputs and outputs can admit contradictions, for example because some components of the service are malfunctioning or subject to malicious manipulation.

Example 1. Let $\ddot{\boldsymbol{G}}$ be the underlying iMTL-algebra, that is weights are a real number in the interval $[0,1]$. Consider the specification SP_0 over signature $\langle\{in, out\}, \varnothing\rangle$ where the set of propositions is empty and the set of actions is Act $= \{in, out\}$ with $\{in\}$ standing for the input of a text file and $\{out\}$ standing for the output of a zip-file. SP_0 is a very loose specification, whose only requirement is $\langle in\rangle\top \wedge [out]\bot$, i.e. at the beginning of a computation only an input action is allowed. Let $P_0 = (W, w_0, R, V)$ be the following process:

$$out|(0.7, 0.4) \quad w_0 \circlearrowright \quad in|(1,1)$$

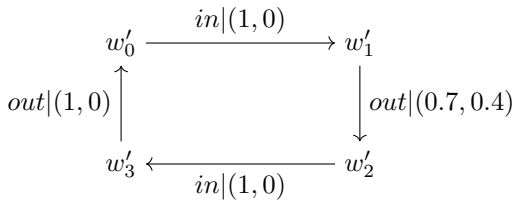
Notice that[1],

$$\begin{aligned}
&Mod(SP_0)(P_0)\\
&= P_0 \models (\langle in\rangle\top \wedge [out]\bot)\\
&= P_0, w_0 \models (\langle in\rangle\top \wedge [out]\bot)\\
&= (P_0, w_0 \models \langle in\rangle\top) \pi (P_0, w_0 \models [out]\bot)\\
&= (\langle in^+\rangle(P_0, w_0, \top^+), [in^+](P_0, w_0, \top^-)) \pi\\
&\quad ([out^+](P_0, w_0, \bot^+), \langle out^+\rangle(P_0, w_0, \bot^-))\\
&= \big(\min(R^+_{in}(w_0, w_0), (P_0, w_0 \models \top)^+), R^+_{in}(w_0, w_0) \to (P_0, w_0 \models \top)^-\big) \pi\\
&\quad \big(R^+_{out}(w_0, w_0) \to (P_0, w_0 \models \bot)^+, \min(R^+_{out}(w_0, w_0), (P_0, w_0 \models \bot)^-)\big)\\
&= (\min(1,1), 1 \to 0) \pi (0.7 \to 0, \min(0.7, 1)) \;=\; (1,0) \pi (0, 0.7)\\
&= (\min(1,0), \max(0, 0.7)) \;= (0, 0.7)
\end{aligned}$$

Since there is some evidence degree that the computation may start with an output action, P_0 has an evidence degree 0 of satisfying the requirements of SP_0 and an evidence degree 0.7 of not satisfying them. Notice that $Mod(SP_0)(P_0)^-$ is equal to the evidence degree of action *out* occurring, that is, $R^+_{out}(w_0, w_0)$.

If we now consider a more concrete specification SP_1 over the same signature, whose requirement is $\downarrow x_0.(\langle in\rangle \downarrow x_1(\langle out\rangle x_0 \wedge [in]\bot) \wedge [out]\bot)$, meaning that at the beginning only an input action is allowed, after every input the next action must be an output action, and after any output action the system must go on with an input returning to the initial state. Consider the following $P_1 = (W', w'_0, R', V')$ process where all transitions represent consistent information except for $R_{out}(w'_1, w'_2)$ that conveys inconsistent information.

[1] Valuation g is omitted since $\langle in\rangle\top \wedge [out]\bot$ is a sentence without free variables.



The requirement of SP_0 is also a requirement of SP_1. Therefore, $SP_0 \rightsquigarrow SP_1$ and $Mod(SP_1)(P_0) = (0,1) \preccurlyeq (0,0.7) = Mod(SP_0)(P_0)$ and $Mod(SP_1)(P_1) = (0,1) \preccurlyeq (1,0) = Mod(SP_0)(P_1)$. As expected, $Mod(SP_1)(P_1) = (0,1)$ since in P_1 after an initial input action followed by an output action the initial state is not reached.

For one specification to implement another (i.e. $SP \rightsquigarrow SP'$) they both need to have the same signature. Such definition of implementation can be too strict since some practices in software development often require implementation decisions, such as introducing new features or reusing previously defined features, which entail the need to deal with different signatures along the development process. The notion of a function called *constructor* that transforms models into other models, possible with different signatures. In such cases *constructor implementations* are the tools to be used.

Given signatures $\Sigma_1, \ldots, \Sigma_n, \Sigma$, a constructor is a function $k : \mathsf{Mod}(\Sigma_1) \times \cdots \times \mathsf{Mod}(\Sigma_n) \rightarrow \mathsf{Mod}(\Sigma)$. For a constructor k and a set of constructors $k_i : \mathsf{Mod}(\Sigma_i^1) \times \cdots \times \mathsf{Mod}(\Sigma_i^{k_i}) \rightarrow \mathsf{Mod}(\Sigma_i)$ for $1 \leqslant i \leqslant n$, the constructor, $k(k_1, \ldots, k_n) : \mathsf{Mod}(\Sigma_1^1) \times \ldots \times \mathsf{Mod}(\Sigma_1^{k_1}) \times \ldots \times \mathsf{Mod}(\Sigma_n^1) \times \ldots \times \mathsf{Mod}(\Sigma_n^{k_n}) \rightarrow \mathsf{Mod}(\Sigma)$ is obtained by the usual composition of functions.

Definition 11. *Let $SP, SP_1, \ldots, SP_n$ be paraconsistent specifications over signatures Σ, $\Sigma_1, \ldots, \Sigma_n$, respectively, and $k : \mathsf{Mod}(\Sigma_1) \times \cdots \times \mathsf{Mod}(\Sigma_n) \rightarrow \mathsf{Mod}(\Sigma)$ a constructor. We say that $(SP_1, \ldots, SP_n)$ is a constructor implementation via k of SP, in symbols $SP \rightsquigarrow_k (SP_1, \ldots, SP_n)$ if for any $P_i \in \mathsf{Mod}(\Sigma_i)$*

$$\prod_{i=1}^{n} Mod(SP_i)(P_i) \preccurlyeq Mod(SP)(k(P_1, \ldots, P_n))$$

The implementation is said to involve decomposition if $n > 1$.

We illustrate the concept of a constructor by redefining the constructors documented in reference [12] to suit our paraconsistent logic.

Definition 12. *Let $\Sigma = \langle \mathrm{Act}, \mathrm{Prop} \rangle$ and $\Sigma' = \langle \mathrm{Act}', \mathrm{Prop}' \rangle$ be signatures such that $\mathrm{Prop} \subseteq \mathrm{Prop}'$ and $\mathrm{Act} \subseteq \mathrm{Act}'$. The signature extension constructor is $k_{ext} : \mathsf{Mod}(\Sigma) \rightarrow \mathsf{Mod}(\Sigma')$. Let $P = (W, w_0, R, V)$ be a process. Then, $k_{ext}(P) = (W, w_0, R', V')$ with*

- $R'_a[w] = \begin{cases} R_a[w] & \text{if } a \in \mathrm{Act} \\ \{(w, w', 0, 1) \text{ for all } w' \in W\} & \text{otherwise} \end{cases}$

– $V'(w,p) = \begin{cases} V(w,p) & \text{if } p \in \text{Prop} \\ (0,1) & \text{otherwise} \end{cases}$

Definition 13. *Let* $\Sigma_1 = \langle \text{Act}, \text{Prop} \rangle$ *and* $\Sigma_2 = \langle \text{Act}', \text{Prop}' \rangle$ *be signatures. The parallel composition constructor is* $k_\otimes : \mathsf{Mod}(\Sigma_1) \times \mathsf{Mod}(\Sigma_2) \to \mathsf{Mod}(\Sigma^\otimes)$ *where* $\Sigma^\otimes = \langle \text{Act} \cup \text{Act}', \text{Prop} \cup \text{Prop}' \rangle$. *The parallel composition of* $P = (W, w_0, R, V)$ *and* $P' = (W', w'_0, R', V')$ *is* $P \otimes P' = (W^\otimes, (w_0, w'_0), R^\otimes, V^\otimes)$ *with*

– $W^\otimes = W \times W'$
– *for any* $(w, w') \in W^\otimes$
  – *if* $a \in \text{Act} \cap \text{Act}'$, $R_a(w,v) = (\alpha, \beta)$ *and* $R'_a(w', v') = (\alpha', \beta')$, *then* $(v, v') \in W^\otimes$ *and* $R^\otimes_a((w,w'),(v,v')) = (\alpha \sqcap \alpha', \beta \sqcup \beta')$
  – *if* $a \in \text{Act} \setminus \text{Act}'$, $R_a(w,v) = (\alpha, \beta)$, *then* $(v, w') \in W^\otimes$ *and* $R^\otimes_a((w,w'),(v,w')) = (\alpha, \beta)$
  – *if* $a \in \text{Act}' \setminus \text{Act}$, $R'_a(w', v') = (\alpha', \beta')$, *then* $(w, v') \in W^\otimes$ *and* $R^\otimes_a((w,w'),(w,v')) = (\alpha', \beta')$
– *for any* $(w, w') \in W^\otimes$
  – *if* $p \in \text{Prop} \cap \text{Prop}'$, $V^\otimes((w,w'),p) = V(w,p) \boxplus V'(w',p)$
  – *if* $p \in \text{Prop} \setminus \text{Prop}'$, $V^\otimes((w,w'),p) = V(w,p)$
  – *if* $p \in \text{Prop}' \setminus \text{Prop}$, $V^\otimes((w,w'),p) = V'(w',p)$

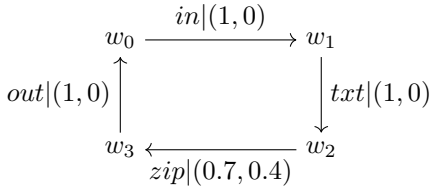
Example 2. A specification interface for SP_1 is now built from two components. One is *Ctrl* with actions $\text{Act}_{Ctrl} = \{in, txt, zip, out\}$. It receives an input, action *in*, from the user, to be given with action *txt* to the other component *GZip*, and receives a zip-file, action *zip*, that is returned with action *out*. This behaviour is specified by

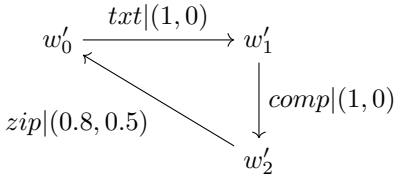
$$\downarrow xo.(\langle in \rangle \downarrow x_1.\big(\langle txt \rangle \downarrow x_2.(\langle zip \rangle \downarrow x_3.(\langle out \rangle x_0 \wedge [-out]\bot) \wedge [-zip]\bot) \wedge [-txt]\bot\big) \wedge [-in]\bot)$$

The other component is *GZip* with actions $\text{Act}_{GZip} = \{txt, comp, zip\}$. First it receives action *txt* from *Ctrl*, then with action *comp* compresses it, and finally delivers a zip-file with action *zip*. This behaviour is specified as

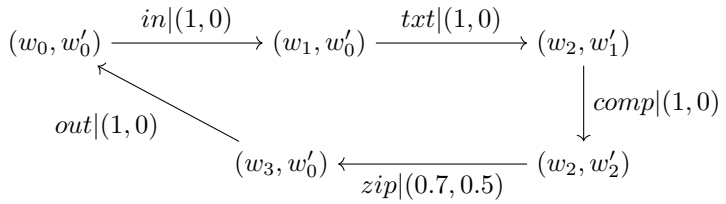
$$\downarrow x_0.(\langle txt \rangle \downarrow x_1.(\langle comp \rangle \downarrow x_2.(\langle zip \rangle x_0 \wedge [-zip]\bot) \wedge [-comp]\bot) \wedge [-txt]\bot)$$

Let P and P' be models of *Ctrl* and *GZip*, respectively.





The parallel composition of process P and P' is the process $P \otimes P'$:



The models of specification $Ctrl \otimes GZip$, by the definition of $\otimes$, consist of all the possible parallel compositions of the models of $Ctrl$ and $GZip$. Therefore, $Ctrl \otimes GZip \rightsquigarrow_{k_\otimes} (Ctrl, GZip)$ is a constructor implementation with decomposition, with the requirements of $Ctrl \otimes GZip$ being similar to the previous ones. Thus,

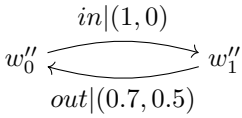
$$\begin{aligned} Mod(Ctrl)(P) \pitchfork Mod(GZip)(P') =& (0.7, 0) \pitchfork (0.8, 0) = (0.7, 0) \\ =& Mod(Ctrl \otimes GZip)(P \otimes P') \end{aligned}$$

Definition 14. *A signature morphism $\sigma : \Sigma \to \Sigma'$ between signatures $\Sigma = \langle \mathrm{Act}, \mathrm{Prop} \rangle$ and $\Sigma' = \langle \mathrm{Act}', \mathrm{Prop}' \rangle$ defines a constructor $k_\sigma : \mathsf{Mod}(\Sigma') \to \mathsf{Mod}(\Sigma)$ that maps any $P' \in \mathsf{Mod}(\Sigma')$ to its reduct $k_\sigma(P') = P'|_\sigma$.*

If σ is bijective then k_σ is a relabelling constructor; if σ is injective then k_σ is a restriction constructor.

Definition 15. *Let $\Sigma = \langle \mathrm{Act}, \mathrm{Prop} \rangle$, $\Sigma' = \langle \mathrm{Act}', \mathrm{Prop}' \rangle$ be signatures and $\Sigma'_D = \langle D, \mathrm{Prop}' \rangle$ also be a signature such that $D \subseteq Str(\mathrm{Act}')$ is a finite subset and $f : \Sigma \to \Sigma'_D$ is a signature morphism. Then, the action refinement constructor $k_f : \mathsf{Mod}(\Sigma'_D) \to \mathsf{Mod}(\Sigma)$ maps any $P' \in \mathsf{Mod}(\Sigma'_D)$ to its reduct $\mathsf{Mod}(f)(P')$.*

Example 3. Let us define an action signature morphism $f : \{in, out\} \to Str(\mathrm{Act}_{Ctrl} \cup \mathrm{Act}_{GZip})$ with $f(in) = in; txt; comp$ and $f(out) = zip; out$. The following process $P = (W, w_0, R, V)$ is the f-reduct of $P \otimes P'$, with $R_{out}(w_1'', w_0'') = R^{\otimes}_{zip;out}((w_2, w_2'), (w_0, w_0')$



We are now able to define an action refinement step $SP_1 \rightsquigarrow_{|_f} Ctrl \otimes GZip$. Notice that, $Mod(Ctrl \otimes GZip)(P \otimes P') = (0.7, 0) = Mod(SP_1)((P \otimes P')|_f)$. Thus, we are now able to define a refinement chain: $SP_0 \rightsquigarrow SP_1 \rightsquigarrow_{|_f} Ctrl \otimes GZip \rightsquigarrow_{k_\otimes} (Ctrl \otimes GZip)$. That can be written as $SP_0 \rightsquigarrow SP_1 \rightsquigarrow_{|_f \circ k_\otimes} (Ctrl, GZip)$.

Often in software development some model does not satisfy exactly the requirements of a specification because of certain implementation details. In this situations a model may still satisfy them abstractly if it exhibits the desired

observable behaviour. Abstractor implementations aim at defining what precisely it means for two models to be identical from an observational perspective. This will be expressed by the use of an equivalence relation $\equiv$ between models, which, as one might expect, comes from a suitable notion of bisimulation between paraconsistent processes. Next definition generalises the bisimulation notion introduced in [3] to multi-modalities:

Definition 16. *Let $\Sigma = \langle \mathrm{Act}, \mathrm{Prop} \rangle$ be a signature and $P = (W, w_0, R, V)$, $P' = (W', w'_0, R', V')$ be Σ-processes. A relation $B \subseteq W \times W'$ is a* bisimulation *between P and P' if for any $(w, w') \in B$:*

(Atom) *for any $p \in \mathrm{Prop}$, $V(w, p) = V'(w', p)$*
(Zig) *for any $v \in W$ such that $R_a(w, v) = (\alpha, \beta)$, there is $v' \in W'$ such that $R'_a(w', v') = (\alpha, \beta)$ and $(v, v') \in B$*
(Zag) *for any $v' \in W'$ such that $R'_a(w', v') = (\alpha, \beta)$, there is $v \in W$ such that $R_a(w, v) = (\alpha, \beta)$ and $(v, v') \in B$*

If there is a bisimulation B such that $(w, w') \in B$ for some $w \in W$, $w' \in W'$, we say that w and w' are bisimilar states and write $w \sim w'$. Given two paraconsistent processes $P = (W, w_0, R, V)$ and $P' = (W', w'_0, R', V')$ over the same signature, we say that P and P' are *behaviourally equivalent*, in symbols $P \equiv P'$, if and only if $w_0 \sim w'_0$. Clearly, $\equiv$ is an equivalence relation.

Let SP be a paraconsistent specification over Σ then, $Mod(\textbf{abstractor } SP)$ is the closure of $Mod(SP)$ under $\equiv$. Thus,

$$Mod(\textbf{abstractor } SP)(M) = \bigsqcup_{N \in [M]_{\equiv}} Mod(SP)(N)$$

where $[M]_{\equiv} = \{N \in \mathsf{Mod}(\Sigma) \mid N \equiv M\}$.

Definition 17. *Let SP, SP' be paraconsistent specifications over Σ and $\equiv \subseteq Mod(\Sigma) \times Mod(\Sigma)$. We write $SP \rightsquigarrow^{\equiv} SP'$ when SP' is a* simple abstractor implementation *of SP, that is, for any $M \in \mathsf{Mod}(\Sigma)$*

$$Mod(SP')(M) \preccurlyeq Mod(\textbf{abstractor } SP)(M)$$

The next definition combines abstractor implementation with constructor for paraconsistent specifications, generalizing [12, Definition 5].

Definition 18. *Let SP, $SP_1, \ldots, SP_n$ be paraconsistent specifications over signatures Σ, $\Sigma_1, \ldots, \Sigma_n$, respectively, $k : \mathsf{Mod}(\Sigma_1) \times \cdots \times \mathsf{Mod}(\Sigma_n) \to \mathsf{Mod}(\Sigma)$ a constructor and $\equiv \subseteq \mathsf{Mod}(\Sigma) \times \mathsf{Mod}(\Sigma)$ an equivalence relation.*
We say that $(SP_1, \ldots, SP_n)$ is an abstractor implementation *of SP, in symbols $SP \rightsquigarrow_k^{\equiv} (SP_1, \ldots, SP_n)$ if for any $P_i \in \mathsf{Mod}(\Sigma_i)$,*

$$\bigsqcap_{i=1}^{n} Mod(SP_i)(P_i) \preccurlyeq Mod(\textbf{abstractor } SP)(k(P_1, \ldots, P_n))$$

Let $k : \mathsf{Mod}(\Sigma_1) \times \cdots \times \mathsf{Mod}(\Sigma_n) \to \mathsf{Mod}(\Sigma)$ be a constructor and, for each $1 \leqslant i \leqslant n$, $\equiv_i$ an equivalence relation between Σ_i-models and $\equiv$ an equivalence relation between Σ-models. We say that a constructor k *preserves abstractions* $\equiv_i$ if for any $M_i, N_i \in \mathsf{Mod}(\Sigma_i)$ such that $M_i \equiv_i N_i$, $k(M_1, .., M_n) \equiv k(N_1, \ldots, N_n)$.

Proposition 2. *The alphabet extension, parallel composition, reduct and action refinement constructors perverse behavioural equivalences, that is,*

- *for any* $P \equiv P'$, $k_{ext}(P) \equiv k_{ext}(P')$
- *for any* $P_1 \equiv P_1'$ *and* $P_2 \equiv P_2'$, $P_1 \otimes P_2 \equiv P_1' \otimes P_2'$
- *let* $\sigma : \Sigma \to \Sigma'$ *be a signature morphism. For any* $P, P' \in \mathsf{Mod}(\Sigma')$, *if* $P \equiv P'$ *then* $P|_\sigma \equiv P'|_\sigma$
- *let* Σ, Σ' *be signatures. Consider the subset* $D \subseteq Str(\mathrm{Act}')$ *and signature* $\Sigma_D = \langle D, \mathrm{Prop}' \rangle$ *and let* $f : \Sigma \to \Sigma_D$ *be a signature morphism. For any* $P, P' \in \mathsf{Mod}(\Sigma_D)$, *if* $P \equiv P'$ *then* $P|_f \equiv P'|_f$

The proof of Proposition 2 is omitted since it is similar to the ones found in [12] for Theorems 4, 5 and 6. Analogously to [12, Theorem 3], next theorem states vertical composition for constructors and abstractor implementations:

Theorem 2. *Consider specifications* $SP, SP_1, \ldots, SP_n$ *over signatures* $\Sigma, \Sigma_1, \ldots, \Sigma_n$ *respectively, a constructor* $k : \mathsf{Mod}(\Sigma_1) \times \cdots \times \mathsf{Mod}(\Sigma_n) \to \mathsf{Mod}(\Sigma)$ *and an equivalence relation* $\equiv \subseteq \mathsf{Mod}(\Sigma) \times \mathsf{Mod}(\Sigma)$ *such that* $SP \rightsquigarrow_k^{\equiv} (SP_1, \ldots, SP_n)$. *For each* $i \in \{1, \ldots, n\}$ *let* $SP_i \rightsquigarrow_{k_i}^{\equiv_i} (SP_i^1, \ldots, SP_i^{r_i})$ *with specifications* $SP_i^1, \ldots, SP_i^{r_i}$ *over signatures* $\Sigma_i^1, \ldots, \Sigma_i^{r_i}$ *respectively, constructors* $k_i : \mathsf{Mod}(\Sigma_i^1) \times \cdots \times \mathsf{Mod}(\Sigma_i^{r_i}) \to \mathsf{Mod}(\Sigma_i)$ *and equivalence relations* $\equiv_i \subseteq \mathsf{Mod}(\Sigma_i) \times \mathsf{Mod}(\Sigma_i)$. *Suppose that* k *preserves the abstractions* $\equiv_i$. *Then,*

$$SP \rightsquigarrow_{k(k_1,\ldots,k_n)}^{\equiv} (SP_1^1, \ldots, SP_1^{r_1}, \ldots, SP_n^1, \ldots, SP_n^{r_n})$$

With $k(k_1, \ldots, k_n) : \mathsf{Mod}(\Sigma_1^1) \times \cdots \times \mathsf{Mod}(\Sigma_1^{r_1}) \times \cdots \times \mathsf{Mod}(\Sigma_n^1) \times \cdots \times \mathsf{Mod}(\Sigma_n^{r_n}) \to \mathsf{Mod}(\Sigma)$ *being a constructor defined by the composition of constructors* k_i *and* k.

Proof. For each $1 \leqslant i \leqslant n$ and for all $1 \leqslant j \leqslant r_i$, let $M_i^j \in \mathsf{Mod}(\Sigma_i^j)$. For each i, by hypothesis, $SP_i \rightsquigarrow_{k_i}^{\equiv_i} (SP_i^1, \ldots, SP_i^{r_i})$,

$$\begin{aligned} \prod_{j=1}^{r_i} Mod(SP_i^j)(M_i^j) &\leqslant Mod(\textbf{abstractor } SP_i \textbf{ w.r.t } \equiv_i)(k_i(M_i^1, \ldots, M_i^{r_i})) \\ &= \coprod_{N_i \in [k_i(M_i^1, \ldots, M_i^{r_i})]_{\equiv_i}} Mod(SP_i)(N_i) \end{aligned}$$

For each i, let M_i be a Σ_i-model such that

$$\begin{aligned} \prod_{j=1}^{r_i} Mod(SP_i^j)(M_i^j) &\leqslant Mod(\textbf{abstractor } SP_i \textbf{ w.r.t } \equiv_i)(k_i(M_i^1, \ldots, M_i^{r_i})) \\ &= Mod(SP_i)(M_i) \end{aligned} \tag{6}$$

By definition of M_i, we have that $M_i \in [k_i(M_i^1, \ldots, M_i^{r_i})]_{\equiv_i}$, that is, $M_i \equiv_i k_i(M_i^1, \ldots, M_i^{r_i})$. Since k preserves abstraction $\equiv_i$,

$$k(M_1, \ldots, M_n) \equiv k(k_1, \ldots, k_n)(M_1^1, \ldots, M_1^{r_1}, \ldots, M_n^1, \ldots, M_n^{r_n}) \tag{7}$$

By hypothesis, $SP \rightsquigarrow_k^{\equiv} (SP_1, \ldots, SP_n)$,

$$\begin{aligned} \prod_{i=1}^{n} Mod(SP_i)(M_i) &\preccurlyeq Mod(\textbf{abstractor } SP \textbf{ w.r.t } \equiv)(k(M_1, \ldots, M_n)) \\ &= \bigsqcup_{N \in [k(M_1, \ldots, M_n)]_{\equiv}} Mod(SP)(N) \end{aligned} \tag{8}$$

By Eq. 7, we have that:

$$k(k_1, \ldots, k_n)(M_1^1, \ldots, M_1^{r_1}, \ldots, M_n^1, \ldots, M_n^{r_n}) \in [k(M_1, \ldots, M_n)]_{\equiv} \tag{9}$$

Since $\equiv$ is transitive, in Eq. 8, $[k(M_1, \ldots, M_n)]_{\equiv}$ can be replaced with $[k(k_1, \ldots, k_n)(M_1^1, \ldots, M_1^{r_1}, \ldots, M_n^1, \ldots, M_n^{r_n})]_{\equiv}$. Considering Eq. 6, we know that for each i,

$$\prod_{j=1}^{r_i} Mod(SP_i^j)(M_i^j) \preccurlyeq Mod(SP_i)(M_i)$$

Since $\prod$ is monotone,

$$\prod_{i=1}^{n} \prod_{j=1}^{r_i} Mod(SP_i^j)(M_i^j) \preccurlyeq \prod_{i=1}^{n} Mod(SP_i)(M_i) \tag{10}$$

With (9) and Eq. 10 we can rewrite Eq. 8 as:
$\prod_{i=1}^{n} \prod_{j=1}^{r_i} Mod(SP_i^j)(M_i^j) \preccurlyeq$
$Mod(\textbf{abst. } SP \textbf{ w.r.t } \equiv)(k(k_1, \ldots, k_n)(M_1^1, \ldots, M_1^{r_1}, \ldots, M_n^1, \ldots, M_n^{r_n}))$
Thus, $SP \rightsquigarrow (SP_1^1 \ldots SP_1^{r_1}, \ldots, SP_n^1, \ldots, SP_n^{r_n})$.

5 Conclusions

This paper is part of on-going research agenda on the (pragmatical) use of paraconsistency in a discipline of software design. Building on previous contributions [3,5] detailed in the Introduction, we define *i)* an institution to frame modelling and reasoning about paraconsistent processes, and *ii)* develop a formal, stepwise development method *à la* Sannella and Tarlecki for this sort of systems. In particular constructor and abstractor implementations were addressed in detail.

There are, of course, several directions for future work. One certainly worth to be explored consists in framing the logics discussed here under the paradigm of asymmetric combination of logics, where the features of a specific logic are developed on top of another one (see e.g. [14]). More precisely, we intend to introduce a systematic way to build paraconsistent modal logics on top of a (base) logic, used to represent the state space. This can be done along the lines of the so-called temporalization [8], and hybridisation of logics [13] processes. A detailed discussion of our method with respect to the approach introduced by Costa [6] to convert classic into paraconsistent logics is also in order.

References

1. Agustí-Cullell, J., Esteva, F., Garcia, P., Godo, L.: Formalizing multiple-valued logics as institutions. In: Bouchon-Meunier, B., Yager, R.R., Zadeh, L.A. (eds.) IPMU 1990. LNCS, vol. 521, pp. 269–278. Springer, Heidelberg (1991). https://doi.org/10.1007/BFb0028112
2. Brauner, T.: Hybrid Logic and its Proof-Theory. Springer, Applied Logic Series (2010)
3. Cruz, A., Madeira, A., Barbosa, L.S.: A logic for paraconsistent transition systems. In: Indrzejczak, A., Zawidzki, M., (eds.) 10th International Conference on Non-Classical Logics. Theory and Applications, vol. 358. EPTCS, pp. 270–284 (2022)
4. Cruz, A., Madeira, A., Barbosa, L.S.: Paraconsistent transition systems. In: Workshop on Logical and Semantic Frameworks, with Applications, EPTCS (in print)
5. Cunha, J., Madeira, A., Barbosa, L.S.: Structured specification of paraconsistent transition systems. In: Fundamentals of Software Engineering. LNCS (in print)
6. de Souza, E.G., Costa-Leite, A., Dias, D.H.B.: On a paraconsistentization functor in the category of consequence structures. J. Appli. Non-Class. Logi. **26**(3), 240–250 (2016)
7. Esteva, F., Godo, L.: Monoidal t-norm based logic: Towards a logic for left-continuous t-norms. Fuzzy Sets Syst. **124**, 271–288 (2001)
8. Finger, M., Gabbay, D.M.: Adding a temporal dimension to a logic system. J. Logic Lang. Inform. **1**(3), 203–233 (1992)
9. Goguen, J.A., Burstall, R.M.: Institutions: Abstract model theory for tpecification and programming. J. ACM **39**(1), 95–146 (1992)
10. Harel, D., Tiuryn, J., Kozen, D.: Dynamic Logic. MIT Press, Cambridge, MA, USA (2000)
11. Kracht, M.: On extensions of intermediate logics by strong negation. J. Philos. Log. **27**(1), 49–73 (1998)
12. Madeira, A., Barbosa, L.S., Hennicker, R., Martins, M.A.: A logic for the stepwise development of reactive systems. Theor. Comput. Sci. **744**, 78–96 (2018)
13. Martins, M.A., Madeira, A., Diaconescu, R., Barbosa, L.S.: Hybridization of institutions. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) CALCO 2011. LNCS, vol. 6859, pp. 283–297. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22944-2_20
14. Neves, R., Madeira, A., Barbosa, L.S., Martins, M.A.: Asymmetric combination of logics is functorial: a survey. In: James, P., Roggenbach, M. (eds.) WADT 2016. LNCS, vol. 10644, pp. 39–55. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-72044-9_4
15. Sannella, D., Tarlecki, A.: Foundations of Algebraic Specification and Formal Software Development. Monographs on TCS, EATCS. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-17336-3