Science of Computer Programming 240 (2025) 103196

Contents lists available at ScienceDirect

Science of Computer Programming

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

Specification of paraconsistent transition systems, revisited [☆]

Juliana Cunha [a,b,*], Alexandre Madeira [a], Luís Soares Barbosa [b]

[a] CIDMA, Dep. Mathematics, Aveiro University, Aveiro, Portugal
[b] INESC TEC & Dep. Informatics, Minho University, Braga, Portugal

ARTICLE INFO

Keywords:
Paraconsistent logic
Specification theory
Dynamic logic
Modal logic

ABSTRACT

The need for more flexible and robust models to reason about systems in the presence of conflicting information is becoming more and more relevant in different contexts. This has prompted the introduction of paraconsistent transition systems, where transitions are characterized by two pairs of weights: one representing the evidence that the transition effectively occurs and the other its absence. Such a pair of weights can express scenarios of *vagueness* and *inconsistency*. This paper establishes a foundation for a compositional and structured specification approach of paraconsistent transition systems, framed as *paraconsistent institution*. The proposed methodology follows the stepwise implementation process outlined by Sannella and Tarlecki.

1. Introduction

In Software Engineering it is often a challenge to cope with modelling contexts in which the classical bivalent logic distinction falls short. This is particularly evident in current quantum computation, especially in NISQ (Noisy Intermediate-Scale Quantum) technology [32], in which levels of decoherence of quantum memory must be articulated with the circuit length to assess program quality. The interested reader is refereed to previous works [14,10], which address this challenge within the paraconsistent framework of this paper. For a brief overview of the problem and the approach discussed see Example 3.

Various modal logics have emerged [6] to address such challenges, specifically aiming to capture vagueness or uncertainty scenarios. Typically, these logics' semantics rely on residuated lattices, which are complete lattices equipped with a commutative monoidal structure such that the monoid composition has a right adjoint, the residuum. The lattice carrier stands for the set of possible truth values. Common examples of such structure are the Boolean set $\{0,1\}$ or the real interval $[0,1]$.

Similarly to the case of fuzzy logic, which resorts to the interval $[0,1]$, the truth values in the paraconsistent logic discussed here represent different degrees of membership, which we will often refer to as degrees of certainty. Unlike probability theory that deals with crisp notions and events that either occur or not, fuzzy logic addresses vagueness in the sense above measuring confidence levels in the occurrence of events.

As an illustration, consider a card game where a special card must remain unplayed for a player to win and it is possible for players to guess who holds the special card with incorrect guesses resulting in defeat. Each player knows their own hand but not others', though they can track played cards and the deck's composition. It is then possible to calculate the probability of each player

☆ This work was financed by PRR - Plano de Recuperação e Resiliência under the Next Generation EU from the European Union within Project Agenda ILLIANCE C644919832-00000035 - Project n 46, as well as by National Funds through FCT, the Portuguese Foundation for Science and Technology, within the project IBEX, with reference PTDC/CCI-COM/4280/2021.

* Corresponding author at: CIDMA, Dep. Mathematics, Aveiro University, Aveiro, Portugal.
E-mail addresses: juliana.cunha@ua.pt (J. Cunha), madeira@ua.pt (A. Madeira), lsb@di.uminho.pt (L.S. Barbosa).

https://doi.org/10.1016/j.scico.2024.103196
Received 15 March 2024; Received in revised form 23 August 2024; Accepted 23 August 2024

Available online 28 August 2024

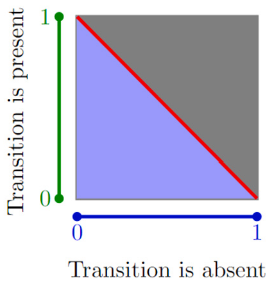

Fig. 1. The vagueness-inconsistency square [13]. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

holding the special card, though this may not be an easy task. In probability theory, the question “Does player 2 have the card?” can be answered with a precise probability. In contrast, fuzzy logic incorporates additional nuances like player behavior and strategies, leading player 1 to believe with some certainty (e.g., 0.2) that player 2 has the card. Unlike the bivalent outcome of probability theory, fuzzy logic deals with statements which can be “more or less true”. For a comprehensive discussion on the differences between fuzzy logic and probability theory see [22].

Since degrees of membership or certainty are not probabilities, belief and disbelief are not always complementary. For example, player 1 might have low certainty, say 0.2, that player 2 has the card due to known bluffing, but high certainty, say 0.9, that player 2 does not have it because another player recently guessed otherwise. As the game progresses, guesses may become more accurate but vague and contradictory beliefs might occur. To handle these scenarios, it is often necessary to extend the underlying Kripke structure by introducing two accessibility relations: one positive and one negative, measuring the certainty of an event occurring or not, respectively. For this purpose, a previous work [13] introduced the concept of a *paraconsistent transition systems*, abbreviated to PLTS. Each transition in these systems is labeled with the possibility of occurring and of failing to do it. The crucial observation is that both relations may carry weights that are not complementary. Informally, the paraconsistent framework discussed in this paper can be seen as a product of fuzzy logics. For an in-depth discussion on how different systems of fuzzy logic can satisfy paraconsistent properties, we refer the interested reader to [20].

Paraconsistent transition systems were introduced in a previous work [13] as a generalization of the structures supporting the Belnap-Dunn four-valued logic [7]. The semantics of the Belnap-Dunn logic is given over the $\mathcal{FOUR}$ lattice depicted below.

$$\begin{tikzcd}
& (1,0) \arrow[dl, no head] \arrow[dr, no head] & \\
(1,1) \arrow[dr, no head] & & (0,0) \arrow[dl, no head] \\
& (0,1) &
\end{tikzcd}$$

in which propositions are interpreted as “true”, “false”, “neither true or false” or as “both true and false”. Consequently, the valuation of propositions is expressed as a pair $(\mathit{tt}, \mathit{ff})$, where $\mathit{tt}, \mathit{ff} \in \{0, 1\}$, with the pairs $(1, 0)$ and $(0, 1)$ representing consistent information. Pair $(1, 1)$ represents inconsistent information, while pair $(0, 0)$ denotes vague information. The relationship between elements in lattice $\mathcal{FOUR}$ follows a truth ordering where $(0, 1) \leq (1, 1) = (0, 0) \leq (1, 0)$. However, other plausible orders of elements can be accommodated [34].

This situation can be generalized. Actually, for paraconsistent transition systems, introduced in a previous work [13], the interpretation of weights resorts to a broader class of residuated lattices over a set A of possible truth values, following a well-known framework [6]. Consequently, all relevant constructions are parametric in a class of residuated lattices, allowing various instances according to the structure of the truth values domain that best suits each modeling problem at hand.

Our previous work [13] extends the approach outlined in other studies [6] to capture paraconsistency, adopting a class of residuated lattices over a set A of possible truth values. Note that in this framework, the classical logic corresponds to the Boolean algebra with two elements. For a general paraconsistent transition system, transitions are then characterized by a pair of weights $(\mathit{tt}, \mathit{ff}) \in A \times A$, of different polarity. For instance, consider the scenario where weights for both transitions are derived from a residuated lattice over the real interval $[0, 1]$. Then, the two accessibility relations jointly express:

- *inconsistency*, when the positive and negative weights are contradictory, i.e. they sum to some value greater than 1 (cf, the upper triangle in Fig. 1 filled in grey).
- *vagueness*, when the sum is less than 1 (cf, the lower, periwinkle triangle in Fig. 1);
- *strict consistency*, when the sum is exactly 1, which means that the measures of the factors enforcing or preventing a transition are complementary, corresponding to the red line in the figure.

Exploring the upper triangle of Fig. 1 calls for paraconsistent logics [24,11], in which inconsistent information is considered as potentially informative. Introduced more than half a century ago, through the pioneering work of F. Asenjo and Newton da Costa, such logics are becoming increasingly popular. Their original focus on mathematical applications has since then expanded, as evidenced by recent literature emphasizing the engineering potential of paraconsistency [3]. Various other domains have also witnessed applications of paraconsistent logic, including robotics [4], quantum logic [12] and quantum computing [1]. In this context, our aim is to address the question: how to specify this sort of paraconsistent behavior in concrete systems?

A structured specification theory for PLTS was proposed in a previous work [16], which this paper extends, by characterizing an institution [21] for paraconsistent transition systems denoted by $\mathrm{L}(\mathcal{A})$. This formalism is parametric in the truth space $\mathcal{A}$, formalized as a metric twisted structure capable of computing pairs of weights. Then, $\mathrm{L}(\mathcal{A})$ becomes a structured specification logic [39], equipped with specific versions of standard structured specification operators *à la CASL* [29]. These results provide formal support for a specification framework for such systems within the established tradition of algebraic specification. Consequently, they offer software engineers formal tools to specify paraconsistent transition systems in a compositional manner.

The notion of a twisted structure, proposed by Kalman [25], arises from the direct product of a lattice with its order-dual. The term “twist” was later coined in Kracht’s paper [26]. Consequently, the resulting lattice carries a natural De Morgan involution, typically interpreted as a form of “negation”. Twisted structures, also known in the literature as twist-products [5] or twist-algebras [33], are a convenient way to represent algebras for the interpretation of non-classical logics. For instance, a well-known result on Nelson algebras, the algebraic counterpart of Nelson paraconsistent logic [30], states that every Nelson algebra is isomorphic to a twist-structure over an Heyting algebra [40,37].

By a metric twisted structure, we mean a twisted structure as constructed in recent works, see [5, Theorem 3.1], enriched with a metric D to compute the distance between two elements of the twisted structure. This enrichment was introduced in the context of paraconsistent transition systems in a previous work [13] and requires the choice of a suitable metric for the underlying lattice. The metric D plays a crucial role as it provides a concrete meaning to the vagueness-inconsistency square depicted in Fig. 1 by enabling a precise definition of each region: consistency, inconsistency and vagueness, see Section 2.2.

Therefore, this paper begins by lifting the residuated structure in which weights take values to a twisted structure [17]. This enhancement involves enriching the twisted structure considered in previous works [13,16] with the residuum property through the addition of an operator $\otimes$ (cf. [5]).

Subsequently, the paper revisits the paraconsistent institution $\mathrm{L}(\mathcal{A})$, introduced in the original conference paper [16], which is parametric in a fixed twisted structure $\mathcal{A}$. A notable difference from the original paper lies in the logical system of $\mathrm{L}(\mathcal{A})$, which is presented here as a modal logical system, wherein Boolean and modal connectives are abbreviated. This offers a clear and more intuitive definition of the logical connectives. Additionally, similarly to another previous work [15], the logical system is enriched with operators from dynamic logic [23], in order to reason about regular modalities of actions and effectively articulate complex and abstract requirements typical of software development processes.

The structured specification framework originally documented [16] is further extended. A formal definition is proposed to convey the concept of simple implementation for paraconsistent specifications, encompassing fundamental studies of horizontal and vertical composition in $\mathrm{L}(\mathcal{A})$. Consequently, this theoretical groundwork paves the way to a methodology of paraconsistent stepwise refinement, facilitating development through a sequence of small, easily comprehensible, and verifiable steps.

Finally, attention is directed towards a theoretical examination of constructor implementation introduced in a previous work for PLTS (with an initial state) [15]. These implementations are considered in software development practices, where implementation decisions often introduce new design features or reuse existing ones, leading to changes in the signatures along the way. A study of vertical constructor implementations is presented, along with the crucial proof that, similarly to the classical case, constructor implementations in the paraconsistent scenario are merely a specific case of simple implementations [39].

In conclusion, we extend the work presented in the original conference paper [16] with three key contributions:

i Rephrasing of the logical system originally defined [16], resulting in a “minimal” modal logical system enriched with dynamic operators.
ii Exploring horizontal and vertical composition for simple paraconsistent specifications.
iii Investigating constructor implementations for paraconsistent specifications, along with their relationship to the previously defined specification-building operators.

Preliminary investigations into constructor implementations for paraconsistent processes, i.e. PLTS with initial states, have been documented in a recent work [15]. However, this paper distinguishes itself by rephrasing the logical system within an appropriate institution. This rephrasing involves a refinement of the logical grammar by adopting standard abbreviations for Boolean connectives, e.g. $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$ and modal connectives, $\langle\alpha\rangle\varphi = \neg[\alpha]\neg\varphi$. Hence, the term “minimal” is used to highlight that these abbreviations are employed to create a cleaner, smaller and less redundant sentence grammar compared to the one used in our previous works [13,15,16]. As documented in recent research [17], this approach is possible by the properties of the twisted structure underlying the paraconsistent institution, which we explore in greater detail in Sections 2.3.3 and 2.3.4. However, such abbreviations are not always applicable in non-classical logics, with Lukasiewicz logic and infinitely many-valued logics serving as notable examples [27,35]. Additionally, the paper expands the study of constructor implementations to paraconsistent systems, rather than processes, presenting further insights and results in this domain.

The remainder of the paper is organized as follows: Section 2 introduces the necessary background definitions essential for the development of the work. Then, Section 3 reconstructs the standard structured specification operators [39] within this institution. It also outlines a formal constructor implementation process inspired by Sannella and Tarlecki’s approach, which aligns with the CASL-like building operators introduced earlier. Moreover, this section details key composition properties for paraconsistent implementations. Finally, Section 4 offers concluding remarks and identifies several lines for future research.

2. An institution for paraconsistent transitions systems

We start by recalling the notion of an institution, followed, in Section 2.2, by a characterization of metric twisted structures which continue the semantic domain upon which the logic is parameterized, as mentioned in the introduction. Such structures amount to a particular class of residuated lattices in which the lattice meet and the monoidal composition coincide, equipped with a metric which entails a concrete meaning to the vagueness-inconsistency square informally described in the introduction. Finally, in Section 2.3, the relevant institution(s) for $\mathrm{L}(\mathcal{A})$ is built in a step by step way and suitably illustrated.

2.1. Institutions

An institution abstractly defines a logical system by specifying the types of signatures, models, and satisfaction relations involved. This framework formalizes various logics, such as Propositional, Equational, First-order, and Higher-order logics [39].

Definition 1 *([21]).* 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 $\Sigma - \mathit{sentences}$ 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 $\Sigma - \mathit{sentences}$ to $\Sigma' - \mathit{sentences}$
- a functor $\mathsf{Mod}_I : \mathsf{Sign}_I^{op} \to \mathbb{C}at$ assigns to each signature Σ the category of $\Sigma - \mathit{models}$. For each signature morphism $\sigma : \Sigma \to \Sigma'$ the functor

  $$\mathsf{Mod}_I(\sigma) : \mathsf{Mod}_I(\Sigma') \to \mathsf{Mod}_I(\Sigma)$$

  translates $\Sigma' - \mathit{models}$ to $\Sigma - \mathit{models}$
- a satisfaction relation $\models_I^\Sigma \subseteq |\mathsf{Mod}_I(\Sigma)| \times \mathsf{Sen}_I(\Sigma)$ determines the satisfaction of $\Sigma - \mathit{sentences}$ by $\Sigma - \mathit{models}$ for each signature $\Sigma \in |\mathsf{Sign}_I|$.

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

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

Graphically,

$$\begin{tikzcd} \Sigma \arrow[d, "\sigma"'] & \mathsf{Mod}_I(\Sigma) \arrow[r, "\models_I^\Sigma"] & \mathsf{Sen}_I(\Sigma) \arrow[d, "\mathsf{Sen}_I(\sigma)"] \\ \Sigma' & \mathsf{Mod}_I(\Sigma') \arrow[u, "\mathsf{Mod}_I(\sigma)"] \arrow[r, "\models_I^{\Sigma'}"] & \mathsf{Sen}_I(\Sigma') \end{tikzcd}$$

Actually, when formalizing multi-valued logics as institutions, the equivalence on the satisfaction condition (1) can be replaced by an equality (cf. [2]):

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

Notation: When clear from context subscripts I and Σ will be omitted. The functor $\mathsf{Mod}(\sigma) : \mathsf{Mod}(\Sigma') \to \mathsf{Mod}(\Sigma)$ is called the σ-reduct functor and often denoted by $-|_\sigma$.

2.2. (Metric) twisted structures

We adopt a similar approach to that presented in a well-known study (see [6]) by focusing our study in a class of residuated lattice over a set A of possible truth values. The transitions of a PLTS, introduced in a previous work [14], are represented by pairs of weights $(tt, ff) \in A \times A$. Here, tt is called the *positive* weight denoted as $(tt, ff)^+$, while ff is called the *negative* weight, denoted as $(tt, ff)^-$. These weights delineate each transition in contrasting manners: one conveys the evidence of its presence, while the other indicates the evidence of its absence.

In this subsection, we delve deeper into the concept of an $\boldsymbol{A}$-twisted structure to manipulate pairs of weights in $A \times A$. The notion of a twisted structure was initially introduced in Kracht's work [26], where it arises from the direct product of the lattice $\boldsymbol{A}$ and its order-dual $\boldsymbol{A}^\partial$. This resulting lattice naturally possesses an involution given by

$$/\!\!/(tt, ff) = (ff, tt)$$

for all tt, $ff \in A$. Various authors have considered expansions of twisted structures with additional properties on the residuated lattice $\boldsymbol{A}$, leading to novel and interesting on twisted structure [8,5,18]. As mentioned earlier in the introduction, this $\boldsymbol{A}$-twisted structure serves as a foundation for manipulating pairs of weights $A \times A$, setting the stage for subsequent discussions in this paper.

Formally, a residuated lattice $\langle A, \sqcap, \sqcup, 1, 0, \odot, \rightarrow, e\rangle$ over a set A is a complete lattice $\langle A, \sqcap, \sqcup, 1, 0\rangle$, equipped with a monoid $\langle A, \odot, e\rangle$ such that $\odot$ has a right adjoint, $\rightarrow$, called the residuum. We will, however, focus on a particular class of residuated lattices in which the lattice meet ($\sqcap$) and monoidal composition ($\odot$) coincide. Thus the adjunction is stated as $a \sqcap b \leq c$ if and only if $b \leq a \rightarrow c$. Since these two operators coincide, we will henceforth omit one when working with the residuated lattices.

Example 1. The following lattices are complete residuated lattices:

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

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

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

We concentrate on *complete residuated lattices* $\mathbf{A}$ whose carrier A supports a metric space (A, d), with a suitable choice of d. Here, $d : A \times A \rightarrow \mathbb{R}^+$ such that $d(x, y) = 0$ if and only if $x = y$, and $d(x, y) \leq d(x, z) + d(z, y)$. This metric is particularly significant as it provides a concrete interpretation for the vagueness-inconsistency square illustrated in Fig. 1. By utilizing this metric d, we can effectively gauge the level of vagueness or inconsistency inherent in a given pair of weights.

Example 2. For each of the complete residuated algebras presented in Example 1, we provide an appropriate choice of d.

i For the Boolean algebra $\mathbf{2}$ a suitable metric is $d(x, y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise} \end{cases}$
ii For the three valued algebra $\mathbf{3}$ a suitable metric is

d	$\bot$	u	$\top$
$\bot$	0	1	2
u	1	0	1
$\top$	2	1	0

iii For the Gödel algebra $\boldsymbol{\ddot{G}}$ a suitable metric is $d(x, y) = \sqrt{(x - y)^2}$.

In order to operate with pairs of truth weights, it was used in a previous work [13] the notion of $\mathbf{A}$-*twisted structure*. This algebraic structure will play a crucial role in the semantics of our institution, consisting of an enrichment of a twist-structure [26] with a metric.

Given a complete residuated lattice $\boldsymbol{A}$, the twist structure over $\boldsymbol{A}$ is obtained by considering the direct product of $\boldsymbol{A}$ and its order-dual. We will consider the definition of product given in other studies [5, Theorem 3.1] enriched with a metric D for pairs of weights.

Definition 2. Given a complete residuated lattice $\boldsymbol{A}$ enriched with a metric d, a $\boldsymbol{A}$-*twisted structure*

$$\mathcal{A} = \langle A \times A, ⩀, \otimes, ⩁, \Rightarrow, /\!\!/, D\rangle$$

is defined for any (a,b), $(c,d) \in A \times A$ as:

- $(a,b) ⩀ (c,d) = (a \sqcap c, b \sqcup d)$
- $(a,b) \otimes (c,d) = (a \sqcap c, (a \to d) \sqcap (c \to b))$
- $(a,b) ⩁ (c,d) = (a \sqcup c, b \sqcap d)$
- $(a,b) \Rightarrow (c,d) = ((a \to c) \sqcap (d \to b), a \sqcap d)$
- $/\!\!/(a,b) = (b,a)$
- $D((a,b),(c,d)) = \sqrt{d(a,c)^2 + d(b,d)^2}$

The order in $\boldsymbol{A}$ is lifted to $\mathcal{A}$ as $(a,b) \preccurlyeq (c,d)$ if and only if ($a \leq c$ and $b \geq d$).

This definition of a twisted structure for operating with pairs of weights differs from the authors' previous work [16,15] due to the inclusion of the conjunction $\otimes$, adopted from [5]. This addition arises because the original $\to$, $\wedge$ adjunction in $\boldsymbol{A}$ does not lift directly to $\mathcal{A}$. Specifically, the following equivalence does not always hold:

$$(a,b) ⩀ (c,d) \preccurlyeq (e,f) \text{ if and only if } (c,d) \preccurlyeq (a,b) \Rightarrow (e,f) \tag{3}$$

For instance, in the Gödel algebra, take $(a,b) = (0.8,0.4)$, $(c,d) = (0.5,0.2)$ and $(e,f) = (0.6,0.3)$. It is evident that:

$$(0.8,0.4) ⩀ (0.5,0.2) = (0.5,0.4) \preccurlyeq (0.6,0.3)$$

However,

$$(0.8,0.4) \Rightarrow (0.6,0.3) = (0.6,0.3) \not\succcurlyeq (0.5,0.2)$$

since $0.3 \not\leq 0.2$. To address this, the $⩀$ operator in (3) is replaced by $\otimes$. Consequently, the operator $\otimes$ has $\Rightarrow$ as its residuum:

$$(a,b) \otimes (c,d) \preccurlyeq (e,f) \text{ if and only if } (a,b) \preccurlyeq (c,d) \Rightarrow (e,f) \tag{4}$$

Further details on the proof are available in a recent work [17].

Finally, it is worth noting that the metric d of the complete residuated lattice extends to a metric D in the twisted structure, defined as:

$$D((a,b),(c,d)) = \sqrt{d(a,c)^2 + d(b,d)^2}$$

This extension allows for the definition [14] of the consistency and paraconsistency sets, denoted by Δ_C and Δ_P, respectively.

$$\Delta_C = \{(a,b) \mid D((a,b),(0,0)) \leq D((a,b),(1,1))\}$$
$$\Delta_P = \{(a,b) \mid D((a,b),(1,1)) < D((a,b),(0,0))\}$$

Δ_C encompasses all pairs within the periwinkle triangle and the red line in Fig. 1, while Δ_P includes pairs located within the upper triangle shaded in grey in Fig. 1. Therefore, by comparing the distance of any pair of weights to the elements $(0,0)$ and $(1,1)$, it is possible to determine whether it represents consistent or inconsistent information. Moreover, the set Δ comprises all pairs equidistant from $(0,0)$ and $(1,1)$, representing strictly consistent information.

$$\Delta = \Delta_C \cap \Delta_P$$

Beyond facilitating the formal delineation of the three regions in Fig. 1, this metric also plays a significant role in the semantics of the logic, particularly in defining the consistency operator $\circ$.

Let us finish this subsection with the following lemma that outlines a few properties concerning pairs of weights, which will be employed subsequently. Readers interested in exploring further properties are referred to [17, Lemma 3.].

Lemma 1. *Let* $\mathcal{A} = \langle A \times A, ⩀, \otimes, ⩁, \Rightarrow, /\!\!/, D\rangle$ *be a* $\boldsymbol{A}$*-twisted structure over a complete residuated lattice* $\boldsymbol{A}$*. Then,*

$$/\!\!/(/\!\!/(a,b) ⩀ /\!\!/(c,d)) = (a,b) ⩁ (c,d) \tag{5}$$

$$/\!\!/\left(\bigsqcap_{i=1}^{n}(a_i,b_i)\right) = \bigsqcup_{i=1}^{n} /\!\!/ (a_i,b_i) \tag{6}$$

$$/\!\!/\Big((a,b) \Rightarrow /\!\!/(c,d)\Big) = (a,b) \otimes (c,d) \tag{7}$$

$$\text{if } (a,b) \preccurlyeq (c,d) \text{ and } (c,d) \preccurlyeq (e,f) \text{ then } (a,b) \preccurlyeq (e,f) \tag{8}$$

$$\textit{if } /\!\!/ (a,b) \Cup (c,d) = (0,1) \textit{ then } (a,b) \Rightarrow (c,d) = (0,1) \tag{9}$$

Proof. The proof of Property (5) is immediate from the definition of $/\!\!/$, $\Cap$ and $\Cup$.

$$\begin{aligned} /\!\!/(/\!\!/(a,b) \Cap /\!\!/(c,d)) &= /\!\!/ ((b,a) \Cap (d,c)) \\ &= /\!\!/ (b \sqcap d, a \sqcup c) \\ &= (a \sqcup c, b \sqcap d) \\ &= (a,b) \Cup (c,d) \end{aligned}$$

The proof of Property (6) follows as,

$$\begin{aligned} /\!\!/ \left(\bigsqcap_{i=1}^{n} (a_i, b_i) \right) &= /\!\!/ \left((a_1,b_1) \Cap (a_2,b_2) \Cap \ldots \Cap (a_n,b_n) \right) \\ &= /\!\!/ (a_1 \sqcap a_2 \sqcap \ldots \sqcap a_n, b_1 \sqcup b_2 \sqcup \ldots \sqcup b_n) \\ &= (b_1 \sqcup b_2 \sqcup \ldots \sqcup b_n, a_1 \sqcap a_2 \sqcap \ldots \sqcap a_n) \\ &= (b_1,a_1) \Cup (b_2,a_2) \Cup \ldots \Cup (b_n,a_n) \\ &= /\!\!/ (a_1,b_1) \Cup /\!\!/(a_2,b_2) \Cup \ldots \Cup /\!\!/(a_n,b_n) \\ &= \bigsqcup_{i=1}^{n} /\!\!/ (a_i,b_i) \end{aligned}$$

The proof of Property (7) follows by definition of the operators from the twisted structure.

$$\begin{aligned} /\!\!/ \left((a,b) \Rightarrow /\!\!/(c,d) \right) &= /\!\!/ \left((a,b) \Rightarrow (d,c) \right) \\ &= /\!\!/ ((a \to d) \sqcap (c \to b), a \sqcap c) \\ &= (a \sqcap c, (a \to d) \sqcap (c \to b)) \\ &= (a,b) \otimes (c,d) \end{aligned}$$

To prove Property (8) assume that $(a,b) \preccurlyeq (c,d)$ and $(c,d) \preccurlyeq (e,f)$. By definition of $\preccurlyeq$,

$$a \leq c \text{ and } b \geq d \text{ and } c \leq e \text{ and } d \geq f$$

Which is equivalent to

$$a \leq c \leq e \text{ and } b \geq d \geq f$$

Hence, $a \leq e$ and $b \geq f$, by the definition of $\preccurlyeq$ that is the same as writing $(a,b) \preccurlyeq (e,f)$. To prove Property (9) note that,

$$/\!\!/(a,b) \Cup (c,d) = (b,a) \Cup (c,d) = (b \sqcup c, a \sqcap d)$$

$$(a,b) \Rightarrow (c,d) = ((a \to c) \sqcap (d \to b), a \sqcap d)$$

By hypothesis, $b \sqcup c = 0$ implies that $b = c = 0$, and $a \sqcap d = 1$ implies that $a = d = 1$. Hence, $(a \to c) \wedge (d \to b) = (1 \to 0) \sqcap (1 \to 0) = 0$ and $a \sqcap d = 1$. Thus, $(a,b) \Rightarrow (c,d) = (0,1)$. ■

2.3. *Framing of* L($\mathcal{A}$) *as an institution*

Let us now fix any given complete residuated lattice $\boldsymbol{A}$ over a non empty set of possible truth values A. As seen in the previous subsection the product $A \times A$ can be endowed with a *twist-structure* as described in Definition 2. The focus now is to properly formalize all the necessary ingredients: signatures, models, sentences and satisfaction relation of a many-valued institution L($\mathcal{A}$) parametric to some fix twisted structure $\mathcal{A}$.

$$\mathrm{L}(\mathcal{A}) = (\mathsf{Sign}, \mathsf{Sen}, \mathsf{Mod}, \models)$$

As mentioned in the introduction, this institution extends the one documented in the original conference paper [16] to incorporate regular modalities of actions, enabling the expression of complex and abstract requirements. While a similar extension was explored in a previous work [15] for paraconsistent processes, i.e. PLTS with initial states, this paper diverges by focusing on systems rather than processes. Additionally, it investigates a simplification of the logical system, resulting in sentences with a smaller, cleaner, and less redundant grammar compared to prior attempts [16,15,13].

2.3.1. Signatures

Definition 3. A signature Σ is a pair (Prop, Act) where Prop is a set of propositions and Act is a set of action symbols. A signature morphism $\sigma : \Sigma \to \Sigma'$ is a pair of functions $\sigma_{\text{Prop}} : \text{Prop} \to \text{Prop}'$ and $\sigma_{\text{Act}} : \text{Act} \to \text{Act}'$.

Signature morphisms can be seen as a change of notation, that is, a change of names of the proposition and action symbols. Morphisms between two signatures are functions between sets such that identities exist and the composition of functions is associative. Hence, signatures and their morphisms form a category called the signature category, denoted by Sign.

The set of atomic actions Act induces a set of *structured actions* [36,28], which is denoted by $Str(\text{Act})$. This set can be regarded as a basic programming language described by the following grammar:

$$\alpha := a \mid \alpha;\alpha \mid \alpha + \alpha \mid \alpha^*$$

where $a \in Act$. A structured action α is formed over atomic actions by using sequential composition (;), nondeterministic choice (+) and iteration (*). This choice of operators is traditional in *regular propositional dynamic logic*, where programs are modeled as regular expressions over atomic symbols. Similarly to the approach taken in dynamic logic [23], in this paper, we aim to combine modalities indexed by structured actions, as detailed in Section 2.3.3.

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

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

$$\begin{tikzcd}
\text{Act} \arrow[r, "\sigma_{\text{Act}}"] \arrow[d, hook] & \text{Act}' \arrow[d, hook] \\
Str(\text{Act}) \arrow[r, "\widehat{\sigma}_{\text{Act}}"] & Str(\text{Act}')
\end{tikzcd}$$

for all $a \in$ Act and $\alpha, \alpha' \in Str(\text{Act})$. Hence, it is possible to inductively define a translation of regular expressions of actions between different signatures.

2.3.2. The models

We define a model in $\text{L}(\mathcal{A})$ as a paraconsistent transition system (PLTS), explored in previous works [16,15]. In these systems, the accessibility and valuation functions are represented by *positive* and *negative* weights, measuring the evidence for a transition occurring or not, and the evidence for a proposition holding or not, respectively.

Definition 4. Let Σ = (Prop, Act) be a signature. A Σ-paraconsistent transition system, is a tuple $M = (W, R, V)$ such that,

- W is a non-empty set of states,
- $R = (R_a : W \times W \to A \times A)_{a \in \text{Act}}$ is an Act-indexed family of total functions, i.e. given any pair of states $(w_1, w_2) \in W \times W$ and an action $a \in$ Act, relation R assigns a pair $(tt, ff) \in A \times A$ such that tt represents the evidence degree of the transition from w_1 to w_2 occurring through action a and ff represents the evidence degree of the transition being prevented from occurring.
- $V : W \times \text{Prop} \to A \times A$ is a valuation function, that assigns to a proposition $p \in$ Prop at a given state w a pair $(tt, ff) \in A \times A$ such that tt is the evidence degree of p holding in w and ff the evidence degree of not holding.

For any pair $(tt, ff) \in A \times A$, $(tt, ff)^+$ denotes the positive weight tt and $(tt, ff)^-$ denotes the negative weight ff.

Previous works [14,10] explore the use of PLTS for modeling quantum circuits and propose a method for representing these circuits as PLTS. The following example briefly illustrates this approach, focusing on the choice of weights in this context.

Example 3. Unlike classical bits, which are either 0 or 1, qubits in quantum computers can exist in a superposition of these states. A challenge in quantum computing is *decoherence*, a phenomenon where superposition states decay to their ground state due to external interference, causing malfunctions if the qubit's coherence time is exceeded. This coherence time is typically given as an interval (worst-case to best-case), indicating how long a qubit stays in superposition.

To model decoherence, a $(\emptyset, \text{Act})$-PLTS is proposed, where Act consists of quantum qubit operations (e.g., Hadamard gate, CNOT gate). In this PLTS, transitions represent qubit operations, where the minimum coherence time determines the negative weight (likelihood of decoherence) and the maximum coherence time determines the positive weight (likelihood of maintaining coherence).

The interpretation of $\alpha \in Str(\text{Act})$ in a model $M = (W, R, V)$ 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 $a \in$ Act

- $\widehat{R}_{\alpha+\alpha'}(w,w') = \widehat{R}_{\alpha}(w,w') \Sqcup \widehat{R}_{\alpha'}(w,w')$
- $\widehat{R}_{\alpha;\alpha'}(w,w') = \bigSqcup_{v\in W} \left(\widehat{R}_{\alpha}(w,v) \Sqcap \widehat{R}_{\alpha'}(v,w')\right)$
- $\widehat{R}_{\alpha^\star}(w,w') = \bigSqcup_{i\geq 0} \widehat{R}^i_{\alpha}(w,w')$. Where,
  - $\widehat{R}^{k+1}_{\alpha}(w,w') = (\widehat{R}^k_{\alpha}; \widehat{R}_{\alpha})(w,w')$
  - $\widehat{R}^0_{\alpha} = \begin{cases} (1,0) & \text{if } w = w' \\ (0,1) & \text{otherwise} \end{cases}$

where $\bigSqcap$ and $\bigSqcup$ are the distributed versions of $\Sqcap$ and $\Sqcup$, respectively.

Example 4. Consider $\mathrm{L}(\ddot{\mathbf{G}})$ and the signature $\Sigma = (\{q\},\{a,b\})$. The following model $M = (W,R,V)$ is a paraconsistent transition system over Σ with $V(w_0,q) = (0,1)$, $V(w_1,q) = (1,0.2)$ and $V(w_2,q) = (0.3,0.3)$.

$$\begin{tikzcd}
w_0 \arrow[r, bend left, "a|(0.3{,}0.4)"] & w_1 \arrow[l, bend left, "b|(0.9{,}1)"] \arrow[r, bend left, "a|(1{,}0.1)"] & w_2 \arrow[l, bend left, "b|(0.2{,}0)"]
\end{tikzcd}$$

Notice that,

$$\widehat{R}_{a+b}(w_1,w_2) = R_a(w_1,w_2) \Sqcup R_b(w_1,w_2) = (1,0.1) \Sqcup (0.2,0) = (1,0)$$

The evidence degree of going from state w_1 to state w_2 through an action a or an action b is 1 while the evidence degree of not being able to go from w_1 to w_2 (neither through action a neither through b) is 0. Note that the pair $(1,0)$ represents consistent information. On the other hand,

$$\begin{aligned}
\widehat{R}_{a;b}(w_1,w_1) &= \bigSqcup_{v\in W} (R_a(w_1,v) \Sqcap R_b(v,w_1)) \\
&= R_a(w_1,w_2) \Sqcap R_b(w_2,w_1) \\
&= (1,0.1) \Sqcap (0.2,0) \\
&= (0.2,0.1) \\
\widehat{R}_{b;a}(w_1,w_1) &= \bigSqcup_{v\in W} (R_b(w_1,v) \Sqcap R_a(v,w_1)) \\
&= R_b(w_1,w_0) \Sqcap R_a(w_0,w_1) \\
&= (0.9,1) \Sqcap (0.3,0.4) \\
&= (0.3,1)
\end{aligned}$$

Thus,

$$\begin{aligned}
\widehat{R}_{(a;b)+(b;a)}(w_1,w_1) &= \widehat{R}_{a;b}(w_1,w_1) \Sqcup \widehat{R}_{b;a}(w_1,w_1) \\
&= (0.2,0.1) \Sqcup (0.3,1) \\
&= (0.3,0.1)
\end{aligned}$$

The evidence degree to transition from w_1 back to w_1 by action a followed by action b $(a;b)$ or by an action b followed by an action a $(b;a)$ of occurring is 0.3 and the evidence degree of being prevented from occurring is 0.1. Note that the pair $\widehat{R}_{(a;b)+(b;a)}(w_1,w_1)$ represents vague information.

Definition 5. Let $M = (W,R,V)$ and $M' = (W',R',V')$ be two $(\mathrm{Prop},\mathrm{Act})$-PLTS. A morphism between M and M' is a function $h : W \to W'$ compatible with the source valuation and transition functions, i.e.

- for each $a \in \mathrm{Act}$, $R_a(w_1,w_2) \preccurlyeq R'_a(h(w_1),h(w_2))$, and
- for any $p \in \mathrm{Prop}$, $w \in W$, $V(w,p) \preccurlyeq V'(h(w),p)$.

We say that M and M' are isomorphic, in symbols $M \cong M'$, whenever there are morphisms $h : M \to M'$ and $h^{-1} : M' \to M$ such that $h' \circ h = id_{W'}$, $h \circ h' = id_W$. $(\mathrm{Prop},\mathrm{Act})$-PLTSs and the corresponding morphisms form a category denoted by Mod, which acts as the model category for our $\mathrm{L}(\mathcal{A})$ logic.

Definition 6. Let $\Sigma = (\text{Prop}, \text{Act})$ be a signature, $\sigma : \Sigma \to \Sigma'$ a signature morphism and $M = (W, R, V)$ a Σ'-PLTS. The *σ-reduct of M* is the $(\text{Prop}, \text{Act})$-PLTS $M|_\sigma = (W|\sigma, R|_\sigma, V|_\sigma)$ such that

- $W|_\sigma = W$,
- for $w, v \in W$ and $a \in \text{Act}$, $(R|_\sigma)_a(w, v) = R_{\sigma(a)}(w, v)$,
- for $p \in \text{Prop}$, $w \in W$, $V|_\sigma(w, p) = V(w, \sigma(p))$.

Lemma 2. *Reducts preserve morphism.*

Proof. Let $\sigma : (\text{Prop}, \text{Act}) \to (\text{Prop}', \text{Act}')$ be a signature morphism and $h : W \to W'$ a morphism between models $M = (W, R, V)$ and $M' = (W', R', V')$. The σ-reduct of M is the model $M|_\sigma = (W|_\sigma, R|_\sigma, V|_\sigma)$, and the σ-reduct of M' is the model $M'|_\sigma = (W'|_\sigma, R'|_\sigma, V'|_\sigma)$. We want to show that there exists a morphism from $M|_\sigma$ to $M'|_\sigma$.

$$\begin{tikzcd}
M = (W, R, V) \arrow[r, "h"] \arrow[d, "-|_\sigma"'] & M' = (W', R', V') \arrow[d, "-|_\sigma"] \\
M|_\sigma = (W|_\sigma, R|_\sigma, V|_\sigma) \arrow[r, "h"] & M'|_\sigma = (W'|_\sigma, R'|_\sigma, V'|_\sigma)
\end{tikzcd}$$

Notice that, by the definition of σ-reduct for any $w \in W$ then $w \in W|_\sigma$, similarly for any $w' \in W'$ then $w' \in W'|_\sigma$. Moreover, a morphism h between states W and W' is a morphism between $W|_\sigma$ and $W'|_\sigma$. We only have to prove that morphism $h : M|_\sigma \to M'|_\sigma$ preserves the source valuation and transition functions. Thus, for each $a \in Act$ and $w, v \in W|_\sigma$

$$\begin{aligned}
(R|_\sigma)_a(w, v) &=_{\{\text{def, of } \sigma\text{-reduct}\}} \\
&\quad R_{\sigma(a)}(w, v) \\
&\preceq_{\{\text{def. of morphism}\}} \\
&\quad R'_{\sigma(a)}(h(w), h(v)) \\
&=_{\{\text{def. of } \sigma\text{-reduct}\}} \\
&\quad (R'|_\sigma)_a(h(w), h(v))
\end{aligned}$$

Similarly, for each $p \in Prop$ and $w, v \in W|_\sigma$

$$\begin{aligned}
V|_\sigma(w, p) &=_{\{\text{def, of } \sigma\text{-reduct}\}} \\
&\quad V(w, \sigma(p)) \\
&\preceq_{\{\text{def. of morphism}\}} \\
&\quad V'(h(w), \sigma(p)) \\
&=_{\{\text{def. of } \sigma\text{-reduct}\}} \\
&\quad V'|_\sigma(h(w), p) \quad \blacksquare
\end{aligned}$$

Hence, each signature morphism $\sigma : (\text{Prop}, \text{Act}) \to (\text{Prop}', \text{Act}')$ defines a functor $\mathsf{Mod}(\sigma) : \mathsf{Mod}(\text{Prop}', \text{Act}') \to \mathsf{Mod}(\text{Prop}, \text{Act})$ that maps systems 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.

2.3.3. The sentences

Once models for $\text{L}(\mathcal{A})$ are characterized, we proceed to define their syntax and the satisfaction relation. Unlike our previous approaches to define a logic for PLTS [13,15,16], we propose a logic whose sentences are given by a smaller grammar to specify properties of paraconsistent structures parametric on a twisted structure $\mathcal{A}$. Therefore, sentence operators are abbreviated similarly to classical propositional and modal logic. This choice is motivated by the fact that abbreviations provide a clear and more intuitive means to reason about paraconsistency. Furthermore, this choice takes into account the intrinsic dualities described in Lemma 1 relative to the operators of the twisted structure, outlined in Definition 2.

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

$$\varphi ::= \bot \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid [\alpha]\varphi \mid \circ\varphi$$

where $p \in Prop$ and $\alpha := a \mid \alpha;\alpha \mid \alpha + \alpha \mid \alpha^*$, for $a \in \mathrm{Act}$.

As in classical propositional logic, we adopt the following abbreviations

$$\top \overset{\text{def}}{=} \neg\bot \qquad\qquad \varphi \vee \varphi' \overset{\text{def}}{=} \neg(\neg\varphi \wedge \neg\varphi')$$

$$\varphi \rhd \varphi' \overset{\text{def}}{=} \neg\varphi \vee \varphi' = \neg(\varphi \wedge \neg\varphi') \qquad\qquad \varphi \bowtie \varphi' \overset{\text{def}}{=} (\varphi \rhd \varphi') \wedge (\varphi' \rhd \varphi)$$

Similarly, as in classical modal logic we adopt the following abbreviation

$$\langle\alpha\rangle\varphi \overset{\text{def}}{=} \neg[\alpha]\neg\varphi$$

Finally, the operator $\circ$ denotes the *consistency operator* traditional in paraconsistent logics. When prefixed to a sentence φ, $\circ$ indicates that φ behaves consistently; in other words, the pair of weights associated with the occurrence or absence of φ lies on the red line or within the periwinkle triangle of Fig. 1. This is equivalent to stating that $\varphi \in \Delta_C$.

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

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

which entails 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.

Finally, with the defined abbreviations, we can establish a sentence translation scheme for operators $\vee$, $\rhd$, $\bowtie$ and $\langle\alpha\rangle$. For instance, consider operator $\langle\alpha\rangle$. We need to verify that

$$\mathsf{Sen}(\sigma)(\langle\alpha\rangle\varphi) = \langle\widehat{\sigma}_{Act}(\alpha)\rangle\,\mathsf{Sen}(\sigma)(\varphi)$$

Using the definition of $\langle\alpha\rangle\varphi$, $\mathsf{Sen}(\sigma)(\langle\alpha\rangle\varphi) = \mathsf{Sen}(\sigma)(\neg[\alpha]\neg\varphi)$, and by the definition of $\mathsf{Sen}(\sigma)$, we obtain

$$\begin{aligned}\mathsf{Sen}(\sigma)(\neg[\alpha]\neg\varphi) &= \neg\mathsf{Sen}(\sigma)([\alpha]\neg\varphi)\\ &= \neg([\widehat{\sigma}_{Act}(\alpha)]\mathsf{Sen}(\sigma)(\neg\varphi))\\ &= \neg([\widehat{\sigma}_{Act}(\alpha)]\neg\mathsf{Sen}(\sigma)(\varphi))\end{aligned}$$

According to the definition of $\langle\alpha\rangle\varphi$, $\neg([\widehat{\sigma}_{Act}(\alpha)]\neg\mathsf{Sen}(\sigma)(\varphi)) = \langle\widehat{\sigma}_{Act}(\alpha)\rangle\mathsf{Sen}(\varphi)$. Similarly, it is possible to verify that the translation scheme defined for the remaining operators $\vee$, $\rhd$ and $\bowtie$ aligns with the definitions provided in a previous work [15].

2.3.4. The satisfaction relation

The satisfaction relation in $\mathrm{L}(\mathcal{A})$ is a function that maps each sentence φ and a PLTS M to a pair of weights $(tt, ff) \in A \times A$. Similar to before, the positive weight tt represents the evidence that φ holds in M, while the negative weight ff signifies the evidence of the opposite fact. It's worth noting that the operators characterizing the twisted structure introduced earlier in Definition 2 resurface in the following satisfaction relation to compute pairs of weights.

Definition 8. Given a signature $\Sigma = (\mathrm{Prop},\mathrm{Act})$, and a Σ-paraconsistent transition system $M = (W, R, V)$, the satisfaction relation

$$\models\, : \mathsf{Mod}(\mathrm{Prop},\mathrm{Act}) \times \mathsf{Sen}(\mathrm{Prop},\mathrm{Act}) \to A \times A$$

is defined by

$$(M \models \varphi) = \bigsqcap_{w \in W} (M, w \models \varphi)$$

where the relation $\models$ is recursively defined as follows:

- $(M, w \models \bot) = (0, 1)$
- $(M, w \models p) = V(w, p)$

- $(M, w \models \neg\varphi) = /\!\!/(M, w \models \varphi)$
- $(M, w \models \varphi \wedge \varphi') = (M, w \models \varphi) \mathbin{⩀} (M, w \models \varphi')$
- $(M, w \models [\alpha]\varphi) = \bigsqcap_{v \in W} \left(\widehat{R}_\alpha(w, v) \Rightarrow (M, v \models \varphi) \right)$
- $(M, w \models \circ\varphi) = \begin{cases} (1,0) & if\ (M, w \models \varphi) \in \Delta_C \\ (0,1) & otherwise \end{cases}$

A sentence φ is said to be *strictly valid* in a PLTS $M = (W, R, V)$ if for any state $w \in W$, $(M, w \models \varphi) = (1, 0)$, i.e. there is complete evidence that φ holds at state w and absolute minimal evidence that it does not hold.

Since modalities are now indexed by regular expressions of actions rather than by atomic actions only, this framework is taken from *dynamic logic* [23]. This extension allows us to express abstract properties [28] such as:

Liveness: $[\text{Act}^*; a]\langle\text{Act}^*; b\rangle\top$ reads "after the occurrence of an action a, an action b can be eventually realised"
Liveness: $[\text{Act}^*; a; (-b)^*]\,\langle\text{Act}^*; b\rangle\top$ reads "after the occurrence of an action a, an occurrence of an action b is eventually possible if it has not occurred before" with $-b$ standing for the set Act $\setminus \{b\}$.
Deadlock avoidance: $[\text{Act}^*]\langle\text{Act}\rangle\top$ reads "after the occurrence of any action, it is possible to read another action".

The approach proposed in this paper aligns with that of recent research [17] but diverges from previous methods [13,16] by introducing abbreviations of sentence connectives. Now, let us revisit the original definition of the satisfaction relation for the abbreviated connectives $\vee$, $\rhd$, and $\langle\alpha\rangle$ which this paper extends, and ensure that these abbreviations are compatible with the initial definition (cf. [16]).

- The sentence $\varphi \vee \varphi'$ was previously defined [16] with an operator $⩁$ defined as the disjunction of pairs of weights, that is,

  $$(M, w \models \varphi \vee \varphi') = (M, w \models \varphi) \mathbin{⩁} (M, w \models \varphi')$$

  This definition is coherent with the abbreviation $\varphi \vee \varphi' = \neg(\neg\varphi \wedge \neg\varphi')$. Since,

  $$\begin{aligned} & (M, w \models \varphi \vee \varphi') \\ =&\{\text{defn, of } \vee\} \\ & (M, w \models \neg(\neg\varphi \wedge \neg\varphi')) \\ =&\{\text{defn, of } \models\} \\ & /\!\!/ \left(/\!\!/ (M, w \models \varphi) \mathbin{⩀} /\!\!/(M, w \models \varphi') \right) \\ =&\{\text{Property (5)}\} \\ & (M, w \models \varphi) \mathbin{⩁} (M, w \models \varphi') \end{aligned}$$

- The sentence $\varphi \rhd \varphi'$ was initially defined [16] with operator $\Rightarrow$ from the twisted structure. That is,

  $$(M, w \models \varphi \rhd \varphi') = (M, w \models \varphi) \Rightarrow (M, w \models \varphi') \tag{10}$$

  In this paper the above definition is replaced by a stronger notion using operators $/\!\!/$ and $⩁$ from the twisted structure. That is,

  $$(M, w \models \varphi \rhd \varphi') = (M, w \models \neg\varphi \vee \varphi') = /\!\!/(M, w \models \varphi) \mathbin{⩁} (M, w \models \varphi') \tag{11}$$

  Similar to what happens in fuzzy logic, the interpretation of the implication operator is not straightforward. In classical bivalent logic, implication $A \to B$ typically binds the consequent B to the antecedent A. This can be defined as $A \to B \overset{\text{def}}{=} \neg A \vee B$, meaning the implication holds whenever A does not hold or B holds. Alternatively, it can be defined through residuation as $A \wedge C \leq B$ if and only if $C \leq A \to B$, where the truth of $A \to B$ is the largest possible truth value C such that $A \wedge C$ is less than or equal to the truth of B [38]. In classical bivalent logic, these two formulas are equivalent. However, in many-valued logics there is no strong reason for the definition of implication to mimic these formulas, especially as the two representations are not necessarily equivalent in these contexts, see [38].
  In this paper, similar to a previous work [17], we propose working with the stronger notion of implication (11). By Property (9), it is possible to formally state that the implication defined in (11) is stronger than (10), that is,

  $$\begin{aligned} &\text{If } /\!\!/ (M, w \models \varphi) \mathbin{⩁} (M, w \models \varphi') = (0, 1) \\ &\quad \text{then } (M, w \models \varphi) \Rightarrow (M, w \models \varphi') = (0, 1) \end{aligned}$$

This comes from the fact that definition (11) seems to better align with natural language, as implication is usually used in its connective sense. That is, in current talk an implication $\varphi \rhd \varphi'$ does not hold whenever φ holds and φ' does not. When dealing with paraconsistency, interpretations often become more complex or even inseparable, making it important to find intuitive notions.
Furthermore, this stronger notion of implication leads to interesting results for equivalence. For instance,

$$(1,1) \bowtie (0,0) = ((1,1) \rhd (0,0)) \mathbin{\sqcap\!\!\!\sqcap} ((0,0) \rhd (1,1)) = (1,0)$$

which interestingly conveys that absolute contradiction $(1,1)$ is equivalent to absolute vagueness $(0,0)$. The same does not occur with the residuated implication, since

$$(1,1) \Leftrightarrow (0,0) = ((1,1) \Rightarrow (0,0)) \mathbin{\sqcap\!\!\!\sqcap} ((0,0) \Rightarrow (1,1)) = (0,0)$$

- Another major difference from previous documentation of logics for PLTS [14,16] lies in the definition of the modal operators $\langle\alpha\rangle\varphi$. Originally, the definition was done by implicitly stating its positive and negative weight. However, in this paper, as in more recent research [17], the definition resorts to the operators from the twisted structure, thus, providing a much cleaner and easier-to-read definition of $\langle\alpha\rangle\varphi$.
  Finally, we want to highlight the importance of the residuum property (4) in the definition of the satisfaction relation for $\langle\alpha\rangle\varphi$. Actually,

$$
\begin{aligned}
(M, w \models \langle\alpha\rangle\varphi) &=_{\{\text{defn. of } \langle\alpha\rangle\varphi\}} \\
&\quad (M, w \models \neg[\alpha]\neg\varphi) \\
&=_{\{\text{defn. of } \models\}} \\
&\quad /\!\!/ \, (M, w \models [\alpha]\neg\varphi) \\
&=_{\{\text{defn. of } \models\}} \\
&\quad /\!\!/ \left(\mathop{\sqcap\!\!\!\sqcap}_{v\in W} \left(\widehat{R}_\alpha(w,v) \Rightarrow (M, v \models \neg\varphi) \right) \right) \\
&=_{\{\text{defn. of } \models\}} \\
&\quad /\!\!/ \left(\mathop{\sqcap\!\!\!\sqcap}_{v\in W} \left(\widehat{R}_\alpha(w,v) \Rightarrow /\!\!/ (M, v \models \varphi) \right) \right) \\
&=_{\{\text{Property (6)}\}} \\
&\quad \mathop{\sqcup\!\!\!\sqcup}_{v\in W} /\!\!/ \left(\widehat{R}_\alpha(w,v) \Rightarrow /\!\!/ (M, v \models \varphi) \right) \\
&=_{\{\text{Property (7)}\}} \\
&\quad \mathop{\sqcup\!\!\!\sqcup}_{v\in W} \left(\widehat{R}_\alpha(w,v) \otimes (M, v \models \varphi) \right)
\end{aligned}
$$

  this definition generalizes the standard definition of $\langle\alpha\rangle\varphi$ in standard bivalent and multi-valued logics.

The following examples serve to illustrate the satisfaction relation in our logic.

Example 5. Consider L(**2**) with the Boolean algebra being the underlying complete residuated lattice, a signature $(\{p\}, \{a\})$ and the PLTS $M = (\{s_0, s_1\}, R, V)$ depicted in the figure below with $V(s_0, p) = (1,0)$ and $V(s_1, p) = (1,1)$.

$$\begin{tikzcd} s_0 \arrow[loop left, "a|(1,1)"] \arrow[r, "a|(1,0)"] & s_1 \end{tikzcd}$$

Notice that,

$$
\begin{aligned}
& M, s_0 \models [a]\,p \\
=& (M, s_0 \models [a]\,p) \mathbin{\sqcup\!\!\!\sqcup} (M, s_0 \models q) \\
=& \mathop{\sqcap\!\!\!\sqcap}_{s\in W} \left(\widehat{R}_a(s_0, s) \Rightarrow (M, s \models p) \right) \\
=& (R_a(s_0, s_0) \Rightarrow (M, s_0 \models p)) \mathbin{\sqcap\!\!\!\sqcap} (R_a(s_0, s_1) \Rightarrow (M, s_1 \models p)) \\
=& ((1,1) \Rightarrow (1,0)) \mathbin{\sqcap\!\!\!\sqcap} ((1,0) \Rightarrow (1,1)) \\
=& (1,0) \mathbin{\sqcap\!\!\!\sqcap} (0,1) \\
=& (1 \sqcap 0, 0 \sqcup 1)
\end{aligned}
$$

$$=(0,1)$$

At state s_0 the sentence $[a]\,p$ holds with evidence degree 0 and doesn't hold with evidence degree 1 so we are in a case where the pair of weights are consistent. Hence, we have consistent evidence that $[a]\,p$ does not hold at state s_0 that is mostly because we have consistent evidence that it is possible to transition through action a to state s_1 however at state s_1 there is complete evidence that p does not hold, i.e. $V^{-}(s_1,p)=1$.

Example 6. Let L(**3**) with the three valued algebra being the underlying complete residuated lattice and $M=(\{s_0,s_1\},R,V)$ be a $(\{p,q,r\},\{a,b\})$-PLTS depicted in the figure below.

$$\begin{tikzcd}
s_0 \arrow[loop left, "b|(\top,u)"] \arrow[r, bend left, "a|(\top,\bot)"] & s_1 \arrow[l, bend left, "b|(\top,\top)"] \arrow[loop right, "b|(u,\top)"]
\end{tikzcd}$$

with the following valuation function

V	p	q	r
s_0	$(\top,\top)$	$(\bot,u)$	(u,u)
s_1	$(\bot,u)$	$(\bot,\bot)$	(u,u)

Note that,

$$\begin{aligned}
M,s_0 \models r \rhd (p\vee q) &= /\!\!/\,(M,s_0\models r) \Cup (M,s_0\models (p\vee q))\\
&= /\!\!/\,V(s_0,r) \Cup (V(s_0,p) \Cup V(s_0,q))\\
&=(u,u)\ \Cup\ (\top,\top) \Cup (\bot,u)\\
&=(u\vee_3 \top \vee_3 \bot, u\wedge_3 \top \wedge_3 u)\\
&=(\top,u)
\end{aligned}$$

At state s_0 the sentence $r\to(p\vee q)$ has complete evidence holding and it's unknown the evidence degree in which it doesn't hold. Also,

$$\begin{aligned}
M,s_1\models \langle b\rangle p =& M,s_1\models /\!\!/[b]\,/\!\!/\,p\\
&= /\!\!/\left(\bigcap_{s\in W}\widehat{R}_b(s_1,s)\Rightarrow /\!\!/(M,s\models p)\right)\\
&= /\!\!/\left(\left(R_b(s_1,s_0)\Rightarrow /\!\!/V(s_0,p)\right)\Cap\left(R_b(s_1,s_1)\Rightarrow /\!\!/V(s_1,p)\right)\right)\\
&= /\!\!/\left(\left((\top,\top)\Rightarrow(\top,\top)\right)\Cap\left((u,\top)\Rightarrow(u,\bot)\right)\right)\\
&= /\!\!/\,(\top\wedge_3\top,\top\vee_3\bot)\\
&=(\top,\top)
\end{aligned}$$

That is, in state s_1 the sentence $\langle b\rangle p$ has evidence degree $\top$ of holding and evidence degree $\top$ of not holding. The pair $(\top,\top)$ represents inconsistent information.

Finally, we provide one more example to emphasize the importance of remembering the paraconsistent logic L($\mathcal{A}$) generalizes models in which all the information is consistent.

Example 7. Let L(**2**) with the Boolean algebra being the underlying complete residuated lattice and $M=(\{s_0,s_1\},R,V)$ be a $(\{p\},\{a,b\})$-PLTS depicted in the figure below with $V(s_0,p)=(1,0)$ and $V(s_1,p)=(0,1)$.

$$\begin{tikzcd}
s_0 \arrow[r, bend left, "a|(1,0)"] & s_1 \arrow[l, bend left, "b|(1,0)"]
\end{tikzcd}$$

Notice that both the transition labels and all the valuations represent consistent information. Since we are in a case where all the information of the model is consistent, we expect that the valuation of $[a;b]p$ at state s_0 represents consistent information. Moreover, we can expect by looking at M that the result of the valuation is going to be $(1,0)$ because it is always possible from w_0 to transition through action a followed by action b and reach a state where p is consistently true, i.e. $(1,0)$. Therefore,

$$\begin{aligned}
M, s_0 \models [a;b]p =& \left(\widehat{R}_{a;b}(s_0,s_0) \Rightarrow (M, s_0 \models p)\right) ⩙ \left(\widehat{R}_{a;b}(s_0,s_1) \Rightarrow (M, s_1 \models p)\right)\\
=& \left((R_a(s_0,s_1) ⩙ R_b(s_1,s_0)) \Rightarrow V(s_0,p)\right)\\
& ⩙ \left((R_a(s_0,s_1) ⩙ R_b(s_1,s_1)) \Rightarrow V(s_1,p)\right)\\
=& \left((1,0) ⩙ (1,0) \Rightarrow (1,0)\right) ⩙ \left(((1,0) ⩙ (0,1)) \Rightarrow (0,1)\right)\\
=& ((1,0) \Rightarrow (1,0)) ⩙ ((0,1) \Rightarrow (0,1))\\
=& (1,0)
\end{aligned}$$

which aligns with our expectations.

The following lemma proves the satisfaction condition (2) for institution $\mathrm{L}(\mathcal{A})$.

Lemma 3. *Let* $\sigma : (\mathrm{Prop}, \mathrm{Act}) \to (\mathrm{Prop}', \mathrm{Act}')$ *be a signature morphism,* M' *a* $(\mathrm{Prop}', \mathrm{Act}')$*-PLTS, and* $\varphi \in \mathsf{Sen}(\mathrm{Prop}, \mathrm{Act})$ *a sentence. Then, for any* $w \in W$*,*

$$\left(M'|_\sigma \models \varphi\right) = \left(M' \models \mathsf{Sen}(\sigma)(\varphi)\right) \tag{12}$$

Proof. According to the definition of $\models$ it is enough to prove that for any $w \in W$,

$$\left(M'|_\sigma, w \models \varphi\right) = \left(M', w \models \mathsf{Sen}(\sigma)(\varphi)\right)$$

By the definition of $\sigma - reduct$. w is in the model $M'|_\sigma = (W, R, V)$ and $M' = (W', R', V')$. To simplify notation we will write $\sigma(p)$ instead of $\sigma_{\mathrm{Prop}}(p)$ for any $p \in \mathrm{Prop}$ and $\sigma(a)$ instead of $\sigma_{\mathrm{Act}}(a)$ for any $a \in \mathrm{Act}$. The proof follows by induction over the structure of sentences.

The case of $\perp$ is trivial, by the definition of $\models$ and Sen, it follows that

$$(M'|_\sigma, w \models \perp) = (0,1) = (M', w \models Sen(\sigma)(\perp))$$

For sentences $p \in \mathrm{Prop}$, one observes that

$$\begin{aligned}
M', w \models \mathsf{Sen}(\sigma)(p) =& \ \{\text{defn of Sen}\}\\
& M', w \models \sigma(p)\\
=& \ \{\text{defn of} \models\}\\
& V'(w, \sigma(p))\\
=& \ \{\text{defn of } \sigma - reduct\}\\
& V(w, p)\\
=& \ \{\text{defn of} \models\}\\
& M'|_\sigma, w \models p
\end{aligned}$$

For sentences $\neg\varphi$ we observe that,

$$\begin{aligned}
M', w \models \mathsf{Sen}(\sigma)(\neg\varphi) =& \ \{\text{defn of Sen}\}\\
& M', w \models \neg\mathsf{Sen}(\sigma)(\varphi)\\
=& \ \{\text{defn of} \models\}\\
& /\!\!/ \ (M', w \models \mathsf{Sen}(\sigma)(\varphi))\\
=& \ \{\text{I.H.}\}\\
& /\!\!/ \ (M'|_\sigma, w \models \varphi)
\end{aligned}$$

$$= \{\text{defn of} \models\}$$

$$M'|_\sigma, w \models \neg\varphi$$

For sentences $\varphi \wedge \varphi'$ the proof follows as,

$$\begin{aligned}
M', w \models \mathsf{Sen}(\sigma)(\varphi \wedge \varphi') &= \{\text{defn of Sen}\}\\
& M', w \models \mathsf{Sen}(\sigma)(\varphi) \wedge \mathsf{Sen}(\sigma)(\varphi')\\
&= \{\text{defn of} \models\}\\
& (M', w \models \mathsf{Sen}(\sigma)(\varphi)) \sqcap (M', w \models \mathsf{Sen}(\sigma)(\varphi'))\\
&= \{\text{I.H.}\}\\
& (M'|_\sigma, w \models \varphi) \sqcap (M'|_\sigma, w \models \varphi')\\
&= \{\text{defn of} \models\}\\
& M'|_\sigma, w \models (\varphi \wedge \varphi')
\end{aligned}$$

The proof for sentences $\circ\,\varphi$ follows as,

$$\begin{aligned}
M', w \models \circ\mathsf{Sen}(\sigma)(\varphi) &= \{\text{defn. of} \models\}\\
& \begin{cases}(1,0) & \text{if } (M', w \models \mathsf{Sen}(\sigma)(\varphi)) \in \Delta_C\\ (0,1) & \text{otherwise}\end{cases}\\
&= \{\text{I.H.}\}\\
& \begin{cases}(1,0) & \text{if } (M'|_\sigma, w \models \varphi) \in \Delta_C\\ (0,1) & \text{otherwise}\end{cases}\\
&= \{\text{defn. of} \models\}\\
& M'|_\sigma, w \models \circ\varphi
\end{aligned}$$

Finally, for modal sentences $[\alpha]\varphi$ the proof follows as,

$$\begin{aligned}
M', w \models \mathsf{Sen}(\sigma)([\alpha]\,\varphi) &= \{\text{defn of Sen}\}\\
& M', w \models [\widehat{\sigma}(\alpha)]\,\mathsf{Sen}(\sigma)(\varphi)\\
&= \{\text{defn of} \models\}\\
& \bigsqcap_{v \in W}\left(\widehat{R}'_{\widehat{\sigma}(\alpha)}(w,v) \Rightarrow (M', v \models \mathsf{Sen}(\sigma)(\varphi))\right)\\
&= \{(\text{step} \star)\}\\
& \bigsqcap_{v \in W}\left(\widehat{(R'|_\sigma)}_\alpha(w,v) \Rightarrow (M'|_\sigma, v \models \varphi)\right)\\
&= \{\text{defn of} \models\}\\
& M'|_\sigma, w \models [\alpha]\,\varphi
\end{aligned}$$

For (step $\star$), we have to observe that for all $v \in W$ and for any $\alpha \in Str(\text{Act})$,

$$\widehat{R}'_{\widehat{\sigma}(\alpha)}(w,v) = \widehat{(R'|_\sigma)}_\alpha(w,v) \tag{13}$$

This can be easily seen by induction over the structure of actions. For atomic actions (13) follows immediately from the definition of σ-reduct,

$$R'_{\sigma(a)}(w,v) = (R'|_\sigma)_a(w,v)$$

For nondeterministic choice of actions $\alpha + \alpha'$,

$$\begin{aligned}
\widehat{R}'_{\sigma(\alpha+\alpha')}(w,v) &= \{\text{defn. of } \widehat{R}\}\\
& \widehat{R}'_{\sigma(\alpha)}(w,v) \sqcup \widehat{R}'_{\sigma(\alpha')}(w,v)\\
&= \{\text{I.H. (13)}\}\\
& \widehat{(R'|_\sigma)}_\alpha(w,v) \sqcup \widehat{(R'|_\sigma)}_{\alpha'}(w,v)
\end{aligned}$$

$$= \{\text{defn. of } \hat{R}\}$$

$$\widehat{(R'|_{\sigma})}_{\alpha+\alpha'}(w, v)$$

For sequential composition of actions $\alpha;\alpha'$,

$$\hat{R}'_{\sigma(\alpha;\alpha')}(w, v) = \{\text{defn. of } \hat{R}\}$$

$$\bigsqcup_{z\in W}\left(\hat{R}'_{\sigma(\alpha)}(w, z) \sqcap \hat{R}'_{\sigma(\alpha')}(z, v)\right)$$

$$= \{\text{I.H. (13)}\}$$

$$\bigsqcup_{z\in W}\left(\widehat{(R'|_{\sigma})}_{\alpha}(w, z) \sqcap \widehat{(R'|_{\sigma})}_{\alpha'}(z, v)\right)$$

$$= \{\text{defn. of } \hat{R}\}$$

$$\widehat{(R'|_{\sigma})}_{\alpha;\alpha'}(w, v)$$

Similarly, it is possible to prove that $\hat{R}'_{\sigma(\alpha^{\star})}(w, v) = \widehat{(R'|_{\sigma})}_{\alpha^{\star}}(w, v)$. ■

The next Theorem is a consequence of this entire subsection where every ingredient of institution $\mathrm{L}(\mathcal{A})$ is properly formalized in terms of category theory and the satisfaction condition, for many-valued institutions, is provided.

Theorem 1. *For a given metric twisted structure $\mathcal{A}$,*

$$\mathrm{L}(\mathcal{A}) = (\mathsf{Sign}, \mathsf{Sen}, \mathsf{Mod}, \models)$$

is an institution.

Such abstraction is necessary to get away from the particular syntax of the logic and to focus on building larger specifications in a structured manner.

3. Specification theory for L($\mathcal{A}$)

In Section 3.1, we delve into a structured specification theory for $\mathrm{L}(\mathcal{A})$. This exploration begins with the introduction of *flat paraconsistent specifications*. Since typically, flat specifications serve as the foundation for constructing new specifications through a composition of operators. These operators, designed for composing specifications, are established within a fixed institution, see [9]. This approach ensures the broad applicability of the theory to a diverse array of logics framed as institutions.

Subsequently, Section 3.2 outlines a framework supporting the systematic, incremental development of software programs from a specification of requirements, as documented by Sannella and Tarlecki [39].

It is noteworthy that the specification theory presented in this section extends classical definitions to encompass an arbitrary institution. Thus, in scenarios involving paraconsistent transition systems where all the information is consistent the classical concepts and outcomes of this chapter coincide the classical definitions, see [9,39].

3.1. Structured specification

Once the institution $\mathrm{L}(\mathcal{A})$ is formalized for paraconsistent systems, it becomes feasible to adapt *CASL*-like specification-building operators to accommodate paraconsistency within a system's description.

Typically, the process starts with flat specifications, comprising a signature and a set of axioms. Subsequent specifications are then constructed through a composition of operators. This framework is motivated by the challenge posed by the increasing complexity of specifications, characterized by a growing number of propositions, action symbols, and axioms. Such complexity often renders reasoning about them more challenging. Consequently, attempting to find models that accurately represent such large and abstract systems can be arduous, if not impossible. Thus, there arises a necessity for larger specifications to be built and expanded from smaller ones using specification-building operators.

Let us start by defining what a *paraconsistent specification* is.

Definition 9. A paraconsistent specification is a pair

$$SP = (Sig(SP), Mod(SP))$$

where $Sig(SP)$ is a signature in Sign and the models of SP is a function

$$Mod(SP) : \mathsf{Mod}(Sig(SP)) \to A \times A.$$

For some model $M \in \mathsf{Mod}(Sig(SP))$ we have that $Mod(SP)(M) = (tt, ff)$, with tt representing the evidence degree of M being a model of SP and the value ff representing the evidence degree of M not being a model of SP.

Traditionally, a *flat specification* is denoted as a pair (Σ, Φ), where Σ represents a signature, Φ denotes a set of axioms, and $Mod(\Sigma, \Phi)$ defines the class of all Σ-models that satisfy the axioms Φ. Hence, a Σ-model is either in or not in $Mod(\Sigma, \Phi)$. This bivalent definition is adjusted in Definition 10 to accommodate paraconsistent reasoning. Hence, PLTS can simultaneously satisfy an axiom with some evidence and not satisfy it with other evidence.

From now one, any mentioned specification is a paraconsistent one.

Definition 10. A *flat paraconsistent specifications* is a pair $SP = \left(\Sigma, \Phi\right)$ such that $\Sigma \in |\mathsf{Sign}|$ is a signature and $\Phi \subseteq \mathsf{Sen}(\Sigma)$ a set of Σ-sentences, often called axioms. Consequently,

- $Sig(SP) = \Sigma$
- $Mod(SP)(M) = \bigsqcap_{\varphi \in \Phi} (M \models \varphi) = \bigsqcap_{\varphi \in \Phi} \left(\bigsqcap_{w \in W} (M, w \models \varphi) \right)$

As stated, flat specifications are a basic tool to build small specifications.

The following specification operator allows us to combine specifications with the restriction that they have to be both over the same signature.

Definition 11. (Union) Given two paraconsistent specifications SP, SP' over the same signature, Σ. Then,

- $Sig(SP \cup SP') = \Sigma$
- $Mod(SP \cup SP')(M) = Mod(SP)(M) \sqcap Mod(SP')(M)$

For instance, if $SP_1 = \langle \Sigma, \Phi_1 \rangle$ and $SP_2 = \langle \Sigma, \Phi_2 \rangle$ are flat specifications then,

$$Mod(\langle \Sigma, \Phi_1 \rangle \cup \langle \Sigma, \Phi_2 \rangle)(M) = Mod(\langle \Sigma, \Phi_1 \cup \Phi_2 \rangle)(M)$$

That is, the evidence of a model M satisfying or not the axioms of two flat specifications SP_1 and SP_2 over the same signature corresponds to the conjunction of $Mod(SP_1)(M)$ and $Mod(SP_2)(M)$.

The following operator is a basic renaming operator

Definition 12. (Translation) Given a Σ-paraconsistent specification SP and a signature morphism $\sigma : \Sigma \to (\mathrm{Prop}', \mathrm{Act}')$. Then,

- $Sig(SP\ \mathbf{with}\ \sigma) = (\mathrm{Prop}', \mathrm{Act}')$
- $Mod(SP\ \mathbf{with}\ \sigma)(M') = Mod(SP)(M'|_\sigma)$, for any $M' \in \mathsf{Mod}(\mathrm{Prop}', \mathrm{Act}')$

The **Translation** operator is particularly important when combined with the **Union** operator. Consider the following inclusion morphisms $\iota : \Sigma \hookrightarrow \Sigma \cup \Sigma'$ and $\iota' : \Sigma' \hookrightarrow \Sigma \cup \Sigma'$. It is possible to define a **Sum** operator that allows us to combine specifications SP and SP' over different signatures:

$$SP + SP' \overset{\mathrm{def}}{=} (SP\ \mathbf{with}\ \iota) \cup (SP'\ \mathbf{with}\ \iota')$$

where,

- $Sig(SP + SP') = \Sigma \cup \Sigma'$
- $Mod(SP + SP')(M) = Mod(SP)(M|_\iota) \sqcap Mod(SP')(M|_\sigma)$, for any $M \in \mathsf{Mod}(\Sigma \cup \Sigma')$

This Sum operator provides a straightforward method to combine specifications over different signatures. However, when a symbol appears in both specifications SP and SP', the combined specification $SP + SP'$ includes only one instance of that symbol. To avoid unintended name clashes, a more sophisticated version of the Sum operator can be employed, as detailed in [39, Page 240].

The following operator is usually used to hide auxiliary action and proposition symbols in the implementation process.

Definition 13. (Hiding) Let SP' be a Σ'-paraconsistent specification and consider a signature morphism $\sigma : (\mathrm{Prop}, \mathrm{Act}) \to \Sigma'$. Then,

- $Sig(SP'\ \mathbf{hide\ via}\ \sigma) = (\mathrm{Prop}, \mathrm{Act})$
- $Mod(SP'\ \mathbf{hide\ via}\ \sigma)(M) = \bigsqcup_{N \in M^\sigma} Mod(SP')(N)$

where $M^\sigma = \{N \in \mathsf{Mod}(\Sigma') \mid N|_\sigma = M\}$, i.e. M^σ is the class of all σ-expansions of M.

The next examples illustrate some of the structured specifications operators defined above.

Example 8. Consider L(**2**) with the Boolean algebra being the underlying complete residuated lattice. Given the signature $\Sigma = (\{p,q\},\{b,c\})$ and the inclusion morphism $\sigma : (\{p,q\},\{b,c\}) \hookrightarrow (\{p,q\},\{a,b,c\})$.

$$\begin{tikzcd}
s_1 \arrow[d, "{c|(1,0)}"] \arrow[dr, bend left, "{a|(1,0)}"] & \\
s_0 \arrow[u, bend left, "{b|(1,1)}"] \arrow[r, "{a|(0,1)}", "{c|(1,1)}"'] & s_2 \arrow[loop right, "{c|(1,0)}"]
\end{tikzcd}$$

Consider the paraconsistent specification $SP = (\Sigma, \Phi)$ where

$$\Phi = \{\langle c\rangle\top,\ \neg(p\wedge q),\ p\to\langle c\rangle\neg q\}$$

Let $M' = (\{s_0,s_1,s_2\},R',V')$ be a $(\{p,q\},\{a,b,c\})$-transition model depicted above where

V'	p	q
s_0	$(1,1)$	$(1,1)$
s_1	$(0,0)$	$(1,1)$
s_2	$(1,0)$	$(0,1)$

The following Σ-model $M'|_\sigma = (W,R,V)$ is the σ-reduct of M':

$$\begin{tikzcd}
s_1 \arrow[d, "{c|(1,0)}"] & \\
s_0 \arrow[u, bend left, "{b|(1,1)}"] \arrow[r, "{c|(1,1)}"'] & s_2 \arrow[loop right, "{c|(1,0)}"]
\end{tikzcd}$$

By the definition of σ-reduct: $W = W'$ and $V(s,r) = V'(s,r)$ for any $s\in W$ and $r\in\{p,q\}$. Then,

– $Sig(SP\ \textbf{with}\ \sigma) = (\{p,q\},\{a,b,c\})$
– $Mod(SP\ \textbf{with}\ \sigma)(M') = Mod(SP)(M'|_\sigma) = \bigsqcap_{\varphi\in\Phi}(M'|_\sigma \models \varphi)$

Notice that,

$$\begin{aligned}
&M'|_\sigma \models \langle c\rangle\top\\
=&\Big(M'|_\sigma, s_0 \models \langle c\rangle\top\Big) \sqcup \Big(M'|_\sigma, s_1 \models \langle c\rangle\top\Big) \sqcup \Big(M'|_\sigma, s_2 \models \langle c\rangle\top\Big)\\
=&\Big(R_c(s_0,s_2)\otimes V(s_2,\top)\Big) \sqcup \Big(R_c(s_1,s_2)\otimes V(s_2,\top)\Big) \sqcup \Big(R_c(s_2,s_2)\otimes V(s_2,\top)\Big)\\
=&(1,0) \sqcap (1,0) \sqcap (1,0)\\
=&(1,0)
\end{aligned}$$

The sentence $\langle c\rangle\top$ can be read as "at any state it is possible to read action c". As expected, this is evaluated at model $M'|_\sigma$ by the consistent pair $(1,0)$. Notice that there is a inconsistent transition, $R_c(s_0,s_2)$, however this transition has positive weight equal to 1. However, if we were to replace $R_c(s_0,s_2) = (1,1)$ to $R_c(s_0,s_2) = (0.8,1)$, then $(M'|_\sigma \models \langle c\rangle\top) = (0.8,0)$. Hence, as the positive weight of the inconsistent transition decreases so does $M'|_\sigma \models \langle c\rangle\top$.

For sentence $\neg(p\wedge q)$:

$$(M'|_\sigma \models \neg(p\wedge q)) = \bigsqcap_{s\in W}\Big(M'|_\sigma, s \models \neg(p\wedge q)\Big)$$

$$
\begin{aligned}
&= \bigsqcap_{s\in W} /\!\!/ \left(M'|_\sigma, s \models (p \wedge q) \right)\\
&= \bigsqcap_{s\in W} /\!\!/ \left((M'|_\sigma, s \models p) \Cap (M'|_\sigma, s \models q) \right)\\
&= \bigsqcap_{s\in W} /\!\!/ \left(V(s,p) \Cap V(s,q) \right)\\
&= /\!\!/ \,(1,1) \Cap /\!\!/(0,1) \Cap /\!\!/(0,1)\\
&=(1,1)
\end{aligned}
$$

The valuation represents inconsistent information and that stems from the fact that the valuation's at s_0 are inconsistent. For sentence $p \to \langle c \rangle \neg q$:

$$
\begin{aligned}
M'|_\sigma, s_0 \models p \to \langle c \rangle \neg q &= /\!\!/ \,(M'|_\sigma, s_0 \models p) \Cup (M'|_\sigma, s_0 \models \langle c \rangle \neg q)\\
&= /\!\!/ \, V(s_0, p) \Cup (R_c(s_0, s_2) \otimes /\!\!/ V(s_2, q))\\
&=(1,1) \Cup ((1,1) \otimes /\!\!/(0,1))\\
&=(1,1) \Cup (1,0)\\
&=(1,0)
\end{aligned}
$$

Similarly, we have that $(M'|_\sigma, s_i \models q \to \langle c \rangle q) = (1,0)$, for $i \in \{1,2\}$. Therefore, $(M'|_\sigma \models p \to \langle c \rangle \neg q) = (1,0)$.

In conclusion,

$$
\begin{aligned}
Mod(SP \textbf{ with } \sigma)(M') =& Mod(SP)(M'|_\sigma)\\
=&(M'|_\sigma \models \langle c \rangle \top) \Cap (M'|_\sigma \models \neg(p \wedge q)) \Cap (M'|_\sigma \models p \to \langle c \rangle \neg q)\\
=&(1,1)
\end{aligned}
$$

The degree of which there is evidence that model M' is a model of SP **with** σ, i.e. specification SP translated via the morphism σ, is 1 and the degree to which there is evidence of M' not being a model of the specification is 1. Notice that in this case we have *inconsistency*, we are completely certain that M' satisfies and does not satisfy the axioms Φ. This contradiction stems mainly from the fact that $(M'|_\sigma \models \neg(p \wedge q)) = (1,1)$

3.2. Formal program development

The preceding subsection outlined a robust and adaptable specification framework. In this context, the pursuit of constructing a paraconsistent transition system M in order to accommodate inconsistencies prompts the concept of a paraconsistent specification SP, which maps $Mod(SP)(M)$ to a pair $(\mathit{tt}, \mathit{ff})$ in the truth space. This pair $(\mathit{tt}, \mathit{ff})$ for a given paraconsistent model M, denotes the degree of evidence supporting and refuting the desired behavior of M, respectively.

This subsection presents a framework that facilitates the gradual development of models from a set of desired requirements, either reinforcing or contradicting each other. As a result, this approach leads to the development of models whose information is multi-valued, often reflecting inherent system malfunctions.

The presented framework generalizes classical concepts of refinement steps, regarded as implementations of one specification by another. Emphasis is placed on compositionality, and genericity, culminating in a methodology of paraconsistent implementations. As in the classical case, this methodology entails development through a sequence of small, comprehensible, and verifiable steps:

$$SP_0 \rightsquigarrow SP_1 \rightsquigarrow \ldots \rightsquigarrow SP_n$$

The implementation process unfolds as a series of stepwise implementations, where successive implementations of a specification iteratively yield more concrete specifications. Let us start with the corresponding formalization.

Definition 14. [15] Given two paraconsistent specifications SP and SP' we say that SP' is a *simple implementation* of SP, in symbols $SP \rightsquigarrow SP'$, if

- $Sig(SP') = Sig(SP) = \Sigma$
- $Mod(SP')(M) \preccurlyeq Mod(SP)(M)$, for all $M \in \mathsf{Mod}(\Sigma)$

This definition conveys that the evidence of a model M satisfying a more concrete specification SP' is lower than that of satisfying the less concrete specification SP. Similarly, the evidence of M not satisfying SP' is greater than that of not satisfying SP.

The definition of a simple implementation guarantees that the correctness of the final outcome of stepwise development can be deduced from the correctness of the individual implementation steps. Furthermore, we prove a result akin to [39, Proposition 7.1.2.], showing that the concept of a simple implementation for paraconsistent specifications extends the corresponding classical notion whenever, for any PLTS M, $Mod(SP)(M)$ is either $(1,0)$ or $(0,1)$.

Proposition 1. *For $i \in \{1,\ldots,n\}$, let SP_i be paraconsistent specifications over signature Σ, such that*

$$SP_0 \rightsquigarrow SP_1 \rightsquigarrow \ldots \rightsquigarrow SP_n$$

Then, for any PLTS $M \in \mathsf{Mod}(\Sigma)$,

$$\text{If } Mod(SP_n)(M) = (1,0) \text{ then } Mod(SP_0)(M) = (1,0)$$

Proof. The proof is immediate from the definition of a simple implementation, Definition (14). By hypothesis, $SP_0 \rightsquigarrow SP_1 \rightsquigarrow \ldots \rightsquigarrow SP_n$. It follows that for any PLTS $M \in \mathsf{Mod}(\Sigma)$,

$$Mod(SP_n)(M) \preccurlyeq \ldots \preccurlyeq Mod(SP_2)(M) \preccurlyeq Mod(SP_0)(M)$$

If $Mod(SP_n)(M) = (1,0)$, there is complete evidence that M satisfies the requirements of SP, and complementary there is no evidence it does not satisfy the requirements, Consequently, $Mod(SP_0)(M) \succcurlyeq (1,0)$, which implies, $Mod(SP_0)(M) = (1,0)$. ■

An indirect way to prove the correctness of the final outcome is to notice that the simple implementation relation is transitive.

Theorem 2. ***(Vertical composition)*** *Let SP_1, SP_2 and SP_3 be paraconsistent specifications over the same signature Σ. If $SP_1 \rightsquigarrow SP_2$ and $SP_2 \rightsquigarrow SP_3$, then $SP_1 \rightsquigarrow SP_3$*

Proof. By hypothesis, $Sig(SP_1) = Sig(SP_2) = Sig(SP_3)$. Thus, we only have left to prove that for any PLTS $M \in \mathsf{Mod}(\Sigma)$,

$$Mod(SP_1)(M) \preccurlyeq Mod(SP_3)(M) \tag{14}$$

By hypothesis $SP_1 \rightsquigarrow SP_2$ and $SP_2 \rightsquigarrow SP_3$, that is,

$$Mod(SP_1)(M) \preccurlyeq Mod(SP_2)(M) \text{ and } Mod(SP_2)(M) \preccurlyeq Mod(SP_3)(M)$$

Given Property (8), since $\preccurlyeq$ is a transitive relation assertion (14) follows immediately. ■

The next Theorem proves that the operators **Union**, **Translation** and **Hiding** are monotone.

Theorem 3. ***(Horizontal composition)*** *Consider paraconsistent specifications over the same signature, such that $SP_1 \rightsquigarrow SP_1'$ and $SP_2 \rightsquigarrow SP_2'$. Then,*

1. $(SP_1 \cup SP_2) \rightsquigarrow (SP_1' \cup SP_2')$
2. $(SP_1 \textbf{ with } \sigma) \rightsquigarrow (SP_1' \textbf{ with } \sigma)$
3. $(SP_1 \textbf{ hide via } \sigma) \rightsquigarrow (SP_1' \textbf{ hide via } \sigma)$

Proof. Let $\Sigma = (\mathrm{Prop}, \mathrm{Act})$ and $\Sigma' = (\mathrm{Prop}', \mathrm{Act}')$ be signatures and SP_i, SP_i' for $i \in \{1,2\}$ be Σ-specifications.

To prove statement 1, by the definition of **Union** it follows immediately

$$Sig(SP_1 \cup SP_2) = Sig(SP_1' \cup SP_2') = \Sigma$$

Also, for any model $M \in \mathsf{Mod}(\Sigma)$

$$\begin{aligned}
Mod(SP_1 \cup SP_2)(M) &=_{\{\text{def. of } \cup\}} \\
&\quad Mod(SP_1)(M) \mathbin{⩀} Mod(SP_2)(M) \\
&\preccurlyeq_{\{\text{hypothesis and } ⩀ \text{ is monotone}\}} \\
&\quad Mod(SP_1')(M) \mathbin{⩀} Mod(SP_2')(M) \\
&=_{\{\text{def. of } \cup\}} \\
&\quad Mod(SP_1' \cup SP_2')(M)
\end{aligned}$$

To prove statement 2, let $\sigma : \Sigma \to \Sigma'$ be a signature morphism. Then, by definition of **Translation**

$$Sig(SP_1 \textbf{ with } \sigma) = Sig(SP_1' \textbf{ with } \sigma) = \Sigma'$$

And for any model $M \in \mathsf{Mod}(\Sigma)$

$$\begin{aligned}
Mod(SP_1 \textbf{ with } \sigma)(M) &=_{\{\text{def. of } \textbf{with } \sigma\}} \\
&\quad Mod(SP_1)(M|_\sigma) \\
&\preceq_{\{SP_1 \rightsquigarrow SP_1'\}} \\
&\quad Mod(SP_1')(M|_\sigma) \\
&=_{\{\text{def. of } \textbf{with } \sigma\}} \\
&\quad Mod(SP_1' \textbf{ with } \sigma)(M)
\end{aligned}$$

To prove statement 3, let $\sigma : \Sigma' \to \Sigma$ be a signature morphism. Then, by definition of **Hiding**

$$Sig(SP_1 \textbf{ hide via } \sigma) = Sig(SP_1' \textbf{ hide via } \sigma) = \Sigma'$$

Also, for any model $M \in \mathsf{Mod}(\Sigma)$:

$$\begin{aligned}
\mathsf{Mod}(SP_1 \textbf{ hide via } \sigma)(M) &=_{\{\text{def. of } \textbf{hide via } \sigma\}} \\
&\quad \left(\biguplus_{N \in M^\sigma} Mod(SP_1)(N) \right) \\
&\preceq_{\{\text{Hypothesis } SP_1 \rightsquigarrow SP_1' \text{ and } \biguplus \text{ is monotone}\}} \\
&\quad \left(\biguplus_{N \in M^\sigma} Mod(SP_1')(N) \right) \\
&=_{\{\text{def. of } \textbf{hide via } \sigma\}} \\
&\quad \mathsf{Mod}(SP_1' \textbf{ hide via } \sigma)(M)
\end{aligned}$$

where $M^\sigma = \{N \in Mod(\Sigma') | N|_\sigma = M\}$. ■

The following example is adapted from another work [28] to suit paraconsistent systems and specifications. A similar case study is explored for paraconsistent processes in a previous work [15].

Example 9. Let **3** be the underlying complete residuated lattice and $(\emptyset, \mathrm{Act})$ a signature where the set of propositions is empty and the set of actions is $\mathrm{Act} = \{in, out\}$ with action in standing for the input of a text file and action out standing for the output of a zip-file.

This example considers a file compressing service working only with text files. Starting with a loose specification SP_0 whose requirements are that at any state:

0.1 $[in]\langle out\rangle\top$, whenever a text file is received for compression there exists an output action of the zip-file
0.2 $[\mathrm{Act}^\star]\langle \mathrm{Act}\rangle\top$, the system should never terminate

Let M_0 be the following paraconsistent model whose file compression service is working poorly due to malfunctions. Consequently, the information regarding the input action is inconsistent and the information regarding the output action is vague.

$$\begin{tikzcd} w \arrow[loop left, "in|(\top,\top)"] \arrow[loop right, "out|(u,u)"] \end{tikzcd}$$

It is possible to check that,

$$(M_0, w \models [in]\langle out\rangle\top) = (M_0, w \models [\mathrm{Act}^\star]\langle \mathrm{Act}\rangle\top) = (u, \bot)$$

Hence, $Mod(SP_0)(M_0) = (u, \bot)$.

As stated, SP_0 is a very loose specification that doesn't demand, for example, that immediately after an output action must come an input action. Because of that we will now consider a new specification. Let SP_1 be a specification over Σ whose requirement is that at any state:

1.1 $[out](\langle in\rangle\top \wedge [out]\bot)$, whenever there is an output action the system must go on with an input

It is possible to check that $Mod(SP_1)(M_0) = (u, \bot)$. Hence, let $SP = SP_0 \cup SP_1$ be the union of both specifications.

$$\begin{aligned}
Mod(SP)(M_0) &= Mod(SP_0 \cup SP_1)(M_0) \\
&= Mod(SP_0)(M_0) \sqcap Mod(SP_1)(M_0) \\
&= (u, \bot) \sqcap (\bot, u) = (\bot, u)
\end{aligned}$$

Note that since SP results from the union of SP_0 and SP_1, both flat specifications. Hence, SP axioms consist of the union of the axioms of SP_0 and SP_1, that is, (0.1) + (0.2) and (1.1).

Furthermore, it is trivial that $SP_0 \rightsquigarrow SP$ and $SP_1 \rightsquigarrow SP$. Using the definition of implementation it follows,

$$Mod(SP)(M_0) \preccurlyeq Mod(SP_0)(M_0) \text{ and } Mod(SP)(M_0) \preccurlyeq Mod(SP_1)(M_0)$$

If we now consider the following PLTS, M_1:

$$\begin{tikzcd}
v_0 \arrow[r, "in|(\top,\bot)"] & v_1 \arrow[d, "out|(u,u)"] \\
v_3 \arrow[u, "out|(\top,\bot)"] & v_2 \arrow[l, "in|(\top,\bot)"]
\end{tikzcd}$$

For model M_1 we have that:

$$\begin{aligned}
Mod(SP_0 \cup SP_1)(M_1) =& Mod(SP_0)(M_1) \Cap Mod(SP_1)(M_1) \\
=& (u,\bot) \Cap (\top,\bot) \\
=& (u,\bot)
\end{aligned}$$

Notice that, $Mod(SP)(M_0) \preccurlyeq Mod(SP)(M_1)$, which conveys that there is a higher evidence degree that M_1 satisfies the requirements of SP and a lower evidence degree that M_1 does not satisfy the requirements of SP, compared to model M_0.

For the remainder of this subsection, we will revisit the concept of *constructor implementation*, initially introduced in a prior work [15] for PLTS with initial states, and proceed to present further results while unifying the works documented in [15] with [16]. This constructor implementations are necessary because the notion of simple implementations, in general, may be too restrictive to capture practical software development practices. In software development, implementation decisions often introduce new design features or reuse already implemented ones, typically involving changes to signatures along the way. The concept of constructor implementation provides the necessary generalization for such practices.

Traditionally, the idea behind constructor implementation of a specification SP consists by using not just one, but several specifications $SP'_1, SP'_2, \dots, SP'_n$ as a basis and applying an n-ary constructor such that for any tuple of models from $SP'_1, SP'_2, \dots, SP'_n$, the construction yields a model satisfying SP. Such an implementation is termed a constructor implementation with decomposition, as it relies on multiple components [39]. These concepts are now extended to accommodate reasoning about many-valued outcomes, which may even include inconsistency.

Let us start by recalling the definition of a constructor.

Definition 15. [28] Given signatures $\Sigma_1, ..., \Sigma_n, \Sigma$, a constructor is a function

$$k : \mathsf{Mod}(\Sigma_1) \times ... \times \mathsf{Mod}(\Sigma_n) \to \mathsf{Mod}(\Sigma)$$

For a constructor k and a set of constructors

$$k_i : \mathsf{Mod}(\Sigma_i^1) \times ... \times \mathsf{Mod}(\Sigma_i^{k_i}) \to \mathsf{Mod}(\Sigma_i)$$

for $1 \le i \le n$. It is possible to obtain the following constructor by the usual composition of functions.

$$k(k_1, ..., k_n) : \mathsf{Mod}(\Sigma_1^1) \times ... \times \mathsf{Mod}(\Sigma_1^{k_1}) \times ... \times \mathsf{Mod}(\Sigma_n^1) \times ... \times \mathsf{Mod}(\Sigma_n^{k_n}) \to \mathsf{Mod}(\Sigma)$$

Several examples of constructors for PLTS with initial states are presented in recent research [15]. Let us refer to one of them to illustrate this concept.

Example 10. A signature morphism $\sigma : \Sigma \to \Sigma'$ defines a constructor $k_\sigma : \mathsf{Mod}(\Sigma') \to \mathsf{Mod}(\Sigma)$ that maps any PLTS $M' \in \mathsf{Mod}(\Sigma')$ to its reduct $k_\sigma(M') = M'|_\sigma$.

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

The following definition recalls the notion of a *constructor implementation* for paraconsistent specifications. Once again, the implementation process suggests that as specifications become more concrete, the evidence degree of a model satisfying the specifications decreases, while the evidence of not satisfying them increases.

Definition 16. [15] Let $SP, SP_1, ..., SP_n$ be paraconsistent specifications over signatures Σ, $\Sigma_1, ..., \Sigma_n$, respectively, and

$$k : \mathsf{Mod}(\Sigma_1) \times ... \times \mathsf{Mod}(\Sigma_n) \to \mathsf{Mod}(\Sigma)$$

a constructor. We say that $(SP_1, ..., SP_n)$ is a constructor implementation via k of SP, in symbols $SP \rightsquigarrow_k (SP_1, ..., SP_n)$ if for any $M_i \in \mathsf{Mod}(\Sigma_i)$

$$\bigsqcap_{i=1}^{n} Mod(SP_i)(M_i) \preccurlyeq Mod(SP)(k(M_1, \dots, M_n))$$

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

The next Lemma proves that constructor implementations are just a special case of simple implementations, since each constructor gives rise to a specification-building operation.

Lemma 4. *Given two paraconsistent specifications SP and SP',*

$$SP \rightsquigarrow_k SP' \textit{ if and only if } SP \rightsquigarrow k(SP')$$

Proof. Let SP and SP' be two paraconsistent specifications over signatures Σ and Σ', respectively. Let $k : \mathsf{Mod}(\Sigma') \to \mathsf{Mod}(\Sigma)$ be a constructor that maps PLTS and their morphisms to the corresponding reducts.

($\Longrightarrow$) Let us assume that $SP \rightsquigarrow_k SP'$. The definition of constructor implementation entails that, for any Σ'-paraconsistent transition system M',

$$Mod(SP')(M') \preccurlyeq Mod(SP)(k(M')) \tag{15}$$

Let us now define a new specification $k(SP')$ such that

- $Sig(k(SP')) = \Sigma$
- $Mod(SP')(M') = Mod(k(SP'))(k(M'))$, for any $M' \in \mathsf{Mod}(\Sigma')$

Trivially, $Sig(k(SP)) = Sig(SP') = \Sigma'$. Using (15) and the definition of $k(SP)$ it follows that,

$$Mod(k(SP'))(k(M')) \preccurlyeq Mod(SP)(k(M'))$$

Let us denote $k(M')$ by M, that is, $Mod(k(SP'))(M) \preccurlyeq Mod(SP)(M)$. Finally, by the definition of a simple implementation it is possible to write $SP \rightsquigarrow k(SP')$.

($\Longleftarrow$) Similarly, let us assume that $SP \rightsquigarrow k(SP')$,

$$Mod(k(SP'))(M) \preccurlyeq Mod(SP)(M) \tag{16}$$

and define the paraconsistent specification SP' such that,

- $Sig(SP') = \Sigma'$
- $Mod(k(SP'))(M) = Mod(SP')(M|_\sigma)$, for any $M \in \mathsf{Mod}(\Sigma)$

Using (16) and the definition of SP' it follows that,

$$Mod(SP')(M|_\sigma) \preccurlyeq Mod(SP)(M)$$

Notice that k is the reduct-constructor defined in Example 10 that maps any PLTS $M \in \mathsf{Mod}(\Sigma)$ to its reduct, that is, $k(M) = M|_\sigma$. Hence,

$$Mod(SP')(k(M)) \preccurlyeq Mod(SP)(M)$$

and by definition of constructor implementation we write $SP \rightsquigarrow_k SP'$. ■

Finally, akin to the case of simple implementations, we demonstrate the slightly stronger property that constructor implementations vertically compose.

Lemma 5. ***(Vertical composition)*** *If $SP_0 \rightsquigarrow_k SP_1$ and $SP_1 \rightsquigarrow_{k'} SP_2$ then $SP \rightsquigarrow_{k \circ k'} SP_2$.*

Proof. Let SP_0, SP_1 and SP_2 be paraconsistent specifications over signatures Σ_0, Σ_1 and Σ_2, respectively. Also let the following functions be constructors,

$$k : \mathsf{Mod}(\Sigma_1) \to \mathsf{Mod}(\Sigma_0) \text{ and } k' : \mathsf{Mod}(\Sigma_2) \to \mathsf{Mod}(\Sigma_1)$$

Since constructors k and k' are functions it is possible to define their composition, $k \circ k' : \mathsf{Mod}(\Sigma_2) \to \mathsf{Mod}(\Sigma_0)$. By hypothesis and by the definition of a constructor implementation, for any $M_2 \in \mathsf{Mod}(\Sigma_2)$

$$Mod(SP_2)(M_2) \preccurlyeq Mod(SP_1)(k'(M_2)) \tag{17}$$

Also by hypothesis and by the definition of a constructor implementation,

$$Mod(SP_1)(k'(M_2)) \preccurlyeq Mod(SP_0)(k(k'(M_2)))$$

which is the same as writing $Mod(SP_1)(k(M_2)) \preccurlyeq Mod(SP_0)((k \circ k')(M_2))$.

Since $\preccurlyeq$ is transitive and by (17), it follows that

$$Mod(SP_2)(M_2) \preccurlyeq Mod(SP_0)((k \circ k')(M_2))$$

Thus, $SP_0 \rightsquigarrow_{k \circ k'} SP_2$. ■

4. Conclusions

Bivalent reasoning is often insufficient to capture part of the complexities present in real-world scenarios. This limitation becomes evident in Software Engineering, which has to deal with several simultaneous requirements, either reinforcing or contradicting each other. Such scenarios usually involve notions of uncertainty or informational conflict that can be conceptualized in paraconsistent reasoning.

Motivated by such challenges, this paper revisits paraconsistent transition systems introduced in a previous work [13] within an institutional framework. Such systems involve two pairs of weights: *positive* and *negative*, each characterizing a transition in opposite ways. One weight represents the evidence of its presence, while the other represents the evidence of its absence, respectively. Such pairs of weights enable capturing *consistent*, *vague* and *inconsistent* information. Weights come from a residuated lattice over a set A of possible truth values. Consequently, all the relevant constructions of PLTS are parametric in a class of residuated lattices, thus accommodating different instances according to the structure of the truth values domain that best suits each concrete application scenario.

This paper begins by extending the work documented in the original conference paper [16], specifically by lifting the residuated structure underlying the (parametric) domain of weights of the twisted structure. Thus, the twisted structure is equipped with the residuum property through the addition of an operator $\otimes$ [5]. This enrichment is necessary to define connectives later within the logical system.

Subsequently, the paper formalizes a paraconsistent institution denoted by $\mathrm{L}(\mathcal{A})$, parametric on a fixed twisted structure $\mathcal{A}$. This institution encompasses a modal logic wherein Boolean and modal connectives are abbreviated. Additionally, as in the authors' previous work [15], the institution is enriched with operators from dynamic logic, enabling reasoning with regular modalities of actions. This enhancement allows the expression of complex and abstract requirements typically appearing in software development.

Section 3, revisits the structured specification approach explored in the original conference paper [16] for engineering PLTS compositions. Subsequently, we outline a framework supporting the incremental development of paraconsistent specifications, including the formal definition of an implementation relationship among specifications. We delve into the classical study of horizontal and vertical composition within $\mathrm{L}(\mathcal{A})$. Finally, we discuss constructor implementation, as introduced in a previous work [15] for PLTS. Constructor implementations offer a generic approach to software development, accommodating the introduction of new design features or the reuse of existing ones, often reprising changes in signatures. Additionally, we investigate vertical constructor composition and establish its relationship with simple implementations in the paraconsistent context.

This paper is a part of an ongoing research agenda focused on the pragmatic use of paraconsistency in the field of software design, building upon previous works [16,15]. Consequently, there are several avenues for future exploration and development.

One significant extension lies in the domain of observational abstraction. Abstractor specifications define an abstraction from the standard semantics of a specification with respect to an observational equivalence relation between algebras. While some investigations on abstractor paraconsistent specifications for PLTS have been documented in recent work [15], the observational relation employed is crisp and therefore unable to capture the nuances arising from vague and contradictory information. This prompts exploration of an observational equivalence relation that is inherently paraconsistent, drawing inspiration from similar work on fuzzy relations [31].

The development of an axiomatic system and proof theory for the paraconsistent modal logic introduced in this paper is also worth to pursue. This work is particularly challenging in the context of many-valued logics due to the complexities involved in extending classical results to non-classical frameworks. Previous studies, such as [19] explored similar topics, specifically through Gödel Kripke models (GK). In [19], the authors introduce a notion of logical consequence for these models, denoted by $\vDash_{\leq GK}$, and explore various results. However, these results only pertain to countable theories where the classical notions of logical consequence $\vDash_{GK}$ and $\vDash_{\leq GK}$ are equivalent. Differently, our research aims to extend such results to the paraconsistent framework presented in this paper. Some preliminary work in this direction includes exploring a notion of graded soundness, as detailed in [17, Section 5.2].

Other current research topics include addressing the challenges posed by various application scenarios of PLTS and their corresponding specification theories, spanning diverse fields from robotics to quantum computation [10].

CRediT authorship contribution statement

Juliana Cunha: Investigation. **Alexandre Madeira:** Investigation. **Luís Soares Barbosa:** Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Juan Carlos Agudelo, Walter Alexandre Carnielli, Paraconsistent machines and their relation to quantum computing, J. Log. Comput. 20 (2) (2010) 573–595.
[2] J. Agustí-Cullell, F. Esteva, P. Garcia, Ll Godo, Formalizing multiple-valued logics as institutions, in: Bernadette Bouchon-Meunier, Ronald R. Yager, Lotfi A. Zadeh (Eds.), Uncertainty in Knowledge Bases, Springer, 1991, pp. 269–278.
[3] Seiki Akama (Ed.), Towards Paraconsistent Engineering, Intelligent Systems Reference Library, vol. 110, Springer, 2016.
[4] Jair Minoro Abe, Cláudio Rodrigo Torres, Germano Lambert-Torres, João Inácio da Silva Filho, Helga Gonzaga Martins, Paraconsistent autonomous mobile robot emmy III, in: Germano Lambert-Torres, Jair Minoro Abe, João Inácio da Silva Filho, Helga Gonzaga Martins (Eds.), Advances in Technological Applications of Logical and Intelligent Systems, Selected Papers from the Sixth Congress on Logic Applied to Technology, LAPTEC 2007, Unisanta, Santa Cecilia University, Santos, Brazil, November 21-23, 2007, in: Frontiers in Artificial Intelligence and Applications, vol. 186, IOS Press, 2007, pp. 236–258.
[5] Manuela Busaniche, Roberto Cignoli, The subvariety of commutative residuated lattices represented by twist-products, Algebra Univers. 71 (2014) 02.
[6] Félix Bou, Francesc Esteva, Lluís Godo, Ricardo Oscar Rodríguez, On the minimum many-valued modal logic over a finite residuated lattice, J. Log. Comput. 21 (5) (10 2009) 739–790.
[7] Nuel D. Belnap, A Useful Four-Valued Logic, Springer, Netherlands, Dordrecht, 1977, pp. 5–37.
[8] Manuela Busaniche, Nikolaos Galatos, Miguel Andrés Marcos, Twist structures and Nelson conuclei, Stud. Log. 110 (4) (2022) 949–987.
[9] Dines Bjorner, Martin Henson, Logics of Specification Languages, Springer, Berlin, Heidelberg, 01 2007.
[10] Luís Soares Barbosa, Alexandre Madeira, Capturing qubit decoherence through paraconsistent transition systems, in: Shigeru Chiba, Youyou Cong, Elisa Gonzalez Boix (Eds.), Companion Proceedings of the 7th International Conference on the Art, Science, and Engineering of Programming, Programming 2023, Tokyo, Japan, March 13-17, 2023, ACM, 2023, pp. 109–110.
[11] Walter Carnielli, Marcelo Esteban Coniglio, Paraconsistent Logic: Consistency, Contradiction and Negation, Springer International Publishing, 2016.
[12] Luisa Maria, Dalla Chiara, Roberto Giuntini, Paraconsistent ideas in quantum logic, Synthese 125 (1–2) (2000) 55–68.
[13] Ana Cruz, Alexandre Madeira, Luís Soares Barbosa, A logic for paraconsistent transition systems, in: Andrzej Indrzejczak, Michal Zawidzki (Eds.), 10th International Conference on Non-Classical Logics. Theory and Applications, in: EPTCS, vol. 358, 2022, pp. 270–284.
[14] Ana Cruz, Alexandre Madeira, Luís Soares Barbosa, Paraconsistent transition systems, in: Daniele Nantes-Sobrinho, Pascal Fontaine (Eds.), Proceedings 17th International Workshop on Logical and Semantic Frameworks with Applications, LSFA 2022, Belo Horizonte, Brazil (hybrid), 23-24 September 2022, in: EPTCS, vol. 376, 2022, pp. 3–15.
[15] Juliana Cunha, Alexandre Madeira, Luís Soares Barbosa, Stepwise development of paraconsistent processes, in: David Cristina, Meng Sun (Eds.), Theoretical Aspects of Software Engineering - 17th International Symposium, TASE 2023, Bristol, UK, July 4-6, 2023, Proceedings, in: Lecture Notes in Computer Science, Springer, 2023, pp. 327–343.
[16] Juliana Cunha, Alexandre Madeira, Luís Soares Barbosa, Structured specification of paraconsistent transition systems, in: Hossein Hojjat, Erika Ábrahám (Eds.), Fundamentals of Software Engineering - 10th International Conference, FSEN 2023, Tehran, Iran, May 4-5, 2023, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 14155, 2023, pp. 1–17.
[17] Juliana Cunha, Alexandre Madeira, Luis Barbosa, Paraconsistent transition structures: compositional principles and a modal logic, submitted for publication to a journal, available here.
[18] Jose Castiglioni, M. Menni, Marta Sagastume, On some categories of involutive centered residuated lattices, Stud. Log. 90 (93–124) (2008) 10.
[19] Xavier Caicedo, Ricardo Oscar Rodríguez, Standard Gödel modal logics, Stud. Log. 94 (2) (2010) 189–214.
[20] Rodolfo Ertola, Francesc Esteva, Tommaso Flaminio, Lluís Godo, Carles Noguera, Exploring paraconsistency in degree-preserving fuzzy logics, in: Javier Montero, Gabriella Pasi, Davide Ciucci (Eds.), Proceedings of the 8th Conference of the European Society for Fuzzy Logic and Technology, EUSFLAT-13, Milano, Italy, September 11-13, 2013, Atlantis Press, 2013.
[21] J.A. Goguen, R.M. Burstall, Institutions: abstract model theory for specification and programming, J. ACM 39 (1) (1992) 95–146.
[22] Petr Hájek, Lluís Godo, Francesc Esteva, Fuzzy logic and probability, CoRR, arXiv:1302.4953 [abs], 2013.
[23] David Harel, Dexter Kozen, Jerzy Tiuryn, Dynamic Logic, MIT Press, 2000.
[24] Stanisław Jaśkowski, Propositional calculus for contradictory deductive systems (communicated at the meeting of March 19, 1948), Stud. Log. 24 (1969) 143–160.
[25] J.A. Kalman, Lattices with involution, Trans. Am. Math. Soc. 87 (1958) 485–491.
[26] Marcus Kracht, On extensions of intermediate logics by strong negation, J. Philos. Log. 27 (1) (1998) 49–73.
[27] Grzegorz Malinowski, Many-Valued Logics, Oxford University Press, New York, 1993.
[28] Alexandre Madeira, Luís S. Barbosa, Rolf Hennicker, Manuel A. Martins, A logic for the stepwise development of reactive systems, Theor. Comput. Sci. 744 (2018) 78–96, Theoretical aspects of computing.
[29] Till Mossakowski, Anne Haxthausen, Donald Sannella, Andrzej Tarlecki, CASL, the common algebraic specification language: semantics and proof theory, Comput. Inform. 22 (2003) 285–321.
[30] David Nelson, Constructible falsity, J. Symb. Log. 14 (1) (1949) 16–26.
[31] Linh Anh Nguyen, Logical characterizations of fuzzy bisimulations in fuzzy modal logics over residuated lattices, Fuzzy Sets Syst. 431 (2022) 70–93.
[32] John Preskill, Quantum computing in the nisq era and beyond, Quantum 2 (Aug 2018) 79.
[33] Umberto Rivieccio, Manuela Busaniche, Nelson conuclei and nuclei: the twist construction beyond involutivity, Stud. Log. (Jan 2024).
[34] Umberto Rivieccio, Achim Jung, Ramon Jansana, Four-valued modal logic: Kripke semantics and duality, J. Log. Comput. 27 (1) (06 2015) 155–199.
[35] Gemma Robles, José M. Méndez, A remark on functional completeness of binary expansions of Kleene's strong 3-valued logic, Log. J. IGPL 30 (1) (07 2020) 21–33.
[36] Igor Sedlár, Finitely-valued propositional dynamic logic, in: Nicola Olivetti, Rineke Verbrugge, Sara Negri, Gabriel Sandu (Eds.), 13th Conference on Advances in Modal Logic, AiML 2020, Helsinki, Finland, August 24-28, 2020, College Publications, 2020, pp. 561–579.
[37] Andrzej Sendlewski, Nelson algebras through Heyting ones: I, Stud. Log. 49 (1) (1990) 105–126.
[38] Philippe Smets, Paul Magrez, Implication in fuzzy logic, Int. J. Approx. Reason. 1 (4) (1987) 327–347.
[39] D. Sannella, A. Tarlecki, Foundations of Algebraic Specification and Formal Software Development, Monographs on TCS, an EATCS Series, Springer, 2012.
[40] D. Vakarelov, Notes on n-lattices and constructive logic with strong negation, Stud. Log. 36 (1) (Mar 1977) 109–125.