Paraconsistent Reasoning for Inconsistency Measurement in Declarative Process Specifications

Carl Corea[a], Isabelle Kuhlmann[b], Matthias Thimm[b], John Grant[c]

[a]*Institute for Information Systems Research, University of Koblenz, Koblenz, Germany*
[b]*Artificial Intelligence Group, University of Hagen, Hagen, Germany*
[c]*Department of Computer Science and UMIACS, University of Maryland, College Park, Maryland, USA*

Abstract

Inconsistency is a core problem in fields such as AI and data-intensive systems. In this work, we address the problem of *measuring* inconsistency in declarative process specifications, with an emphasis on linear temporal logic (LTL). As we will show, existing inconsistency measures for classical logic cannot provide a meaningful assessment of inconsistency in LTL in general, as they cannot adequately handle the temporal operators. We therefore propose a novel paraconsistent semantics for LTL over fixed traces (LTL_{ff}) as a framework for time-sensitive inconsistency measurement. We develop and implement novel approaches for (element-based) inconsistency measurement, and propose a novel semantics for reasoning in LTL_{ff} in the presence of preference relations between formulas. We implement our approach for inconsistency measurement with Answer Set Programming and evaluate our results with real-life data sets from the Business Process Intelligence Challenge.

Keywords: Declarative Process Specifications, LTL_f, Inconsistency Measurement

1. Introduction

Linear temporal logic (LTL) is an important logic for specifying the (temporal) behavior of business processes in the form of *declarative process specifications* [1, 2]. The underlying idea is that time is represented as a linear sequence of states $T = (t_0, ..., t_m)$, where t_0 is the designated starting point. At every state, some statements may be true. Temporal operators specify properties that must hold over the sequence of states. For example, the operator **X** (*next*) means that a certain formula holds at the next state. Likewise, the operator **G** (*globally*) means that a certain formula will hold for all following states.

Traditionally, model checking has been used to verify that a particular model—that is, the assignment of truth values for statements over the time sequence—satisfies the requirements. However, a problem in this use-case arises if the set of formulas is *inconsistent*, i. e., contains contradictory specifications. This is a core problem for data-intensive systems, where inconsistencies can easily arise, e. g., due to (modeling) errors within the collected data [3, 4]. For example, consider the two sets of LTL formulas $\mathcal{K}_1$ and $\mathcal{K}_2$:

$$\mathcal{K}_1 = \{\mathbf{X}a, \mathbf{X}\neg a\} \qquad\qquad \mathcal{K}_2 = \{\mathbf{G}a, \mathbf{G}\neg a\}$$

Both $\mathcal{K}_1$ and $\mathcal{K}_2$ are inconsistent, as they demand that both a and $\neg a$ hold in (some) following state, which is unsatisfiable. In such a case, the set of specifications cannot be applied for its intended purpose of process verification. This calls for the analysis of such inconsistencies, to provide insights for inconsistency resolution.

In classical logic, all inconsistent sets are equally bad. However, considering again the two sets, intuitively, $\mathcal{K}_2$ is "more" inconsistent than $\mathcal{K}_1$: The inconsistency in $\mathcal{K}_1$ only affects the next state, while the inconsistency in $\mathcal{K}_2$ affects all following states. This is an important insight that could prove useful for debugging or re-modeling LTL specifications. While there have been some recent works that can *identify* inconsistent sets in declarative process specifications [4, 5, 3], those works cannot look "into" those sets or compare them. In this work, we therefore show how to distinguish the *severity* of inconsistencies in LTL, specifically, a variant of LTL which we coin linear temporal logic on fixed traces (LTL_{ff}).

The scientific field geared towards the quantitative assessment of inconsistency in knowledge representation formalisms is *inconsistency measurement* [6, 7]. Inconsistency measurement studies measures that aim to assess a *degree* of inconsistency with a numerical value. The intuition here is that a higher value represents a higher degree of inconsistency. Such measures can provide valuable insights for debugging inconsistent specifications, e. g., to determine whether certain sets of formulas are more inconsistent than others, or pin-pointing those formulas highly responsible for the overall inconsistency. As we will show, existing measures are currently not geared towards LTL and temporal operators, and therefore cannot provide a meaningful analysis.

Therefore, the main goal of this work is to develop new means for measuring inconsistency in linear temporal logic.

In a previous work [8], we have presented an initial paraconsistent semantics for LTL_{ff}, as well as two baseline inconsistency measures. These results are outlined in Section 3), in particular, we motivate the need for time-sensitive inconsistency measurement (by means of a novel rationality postulate) and define a paraconsistent semantics for LTL_{ff}. Our additional contributions for the present work are as follows.

- In Section 4, we first revisit baseline approaches for measuring inconsistency in LTL_{ff}, including also an evaluation of formal aspects of the introduced measures such as *expressivity*, i. e., the number of distinct values a measure can attain w.r.t. m time points. Then we extend the formalism of LTL_{ff} to be able to consider *preference relations*. In essence, these are relations among the set of formulas, expressing that certain formulas may be preferred over others. As we will show, being able to express preferences can be very valuable for modellers to define exceptions, however, it directly affects the notion of what should be considered as a "consistent" set of formulas. For example, if we consider $\{\mathbf{G}\neg a, \mathbf{X}a\}$ and assume $\mathbf{X}a$ overrules $\mathbf{G}\neg a$, then this specification should be viewed as consistent ($\mathbf{X}a$ is seen as an exception to $\mathbf{G}\neg a$). To enable such forms of reasoning, we develop a new model-theoretic semantics for defeasible LTL_{ff} based on taggings. We show how this semantics can be used to reason over LTL_{ff} with preference relations, and propose initial analysis measures for knowledge bases with preference relations, e. g., by assessing the number of time points where exceptions occur.

- As an extension to inconsistency measures that assess knowledge bases as a whole, we develop element-based measures that can be used for pinpointing the concrete formulas responsible for the overall inconsistency. This provides valuable insights in the scope of inconsistency resolution, e. g., as a basis for re-modeling inconsistent specifications. To guide the development of such measures, we propose a time-sensitivity postulate for element-based measures and show that a Shapley-based approach to element-based measurement satisfies (amongst others) this postulate. Also, we propose new element-based measures for analyzing individual preference relations, e. g., assessing the impact of deleting individual preferences on the overall inconsistency.

- As an application example, in Section 5, we show how our approach can be applied for measuring inconsistency in declarative process specifications, in particular, for the Declare modeling standard. We discuss how our semantics can be applied to the "simple trace" notion in Declare, and present concrete usage examples of our results, in particular, also introducing and extended definition of declarative process models including preference information among constraints, which allows to model exceptions in Declare in a flexible manner.

- Finally, in Section 6, we discuss algorithmic aspects of measuring inconsistency in LTL and implement our approach. Furthermore, we evaluate these implemented results with real-life data sets of the Business Process Intelligence Challenge[1] in Section 7. Our implementation is made available open-source[2] and can be used to assess or compare inconsistency in declarative process specifications. In the context of evaluation, we also investigate the computational complexity of central aspects regarding inconsistency measurement in $\mathrm{LTL}_{\mathrm{ff}}$.

Our investigation is based on preliminaries in Section 2 and is concluded in Section 8. Proofs for all technical results can be found in the appendix.

As stated above, an earlier version of this work has already been published in [9]. This work is a direct extension, where we contribute the contents outlined above. Parts of Sections 2 and 3 are taken from [9].

2. Preliminaries

The traditional setting for inconsistency measurement is that of propositional logic. For that, let At be some fixed propositional signature, i. e., a (possibly infinite) set of propositions, and let $\mathcal{L}(\mathsf{At})$ be the corresponding propositional language constructed using the usual connectives $\wedge$ (*conjunction*), $\vee$ (*disjunction*), and $\neg$ (*negation*). A literal is a proposition p or negated proposition $\neg p$.

Definition 1. *A knowledge base $\mathcal{K}$ is a finite set of formulas $\mathcal{K} \subset \mathcal{L}(\mathsf{At})$. Let $\mathbb{K}$ be the set of all knowledge bases.*

[1]`https://icpmconference.org/2020/bpi-challenge/`

[2]`https://github.com/aig-hagen/inconsistency-measurement-LTL`

For a set of formulas X we denote the set of propositions in X by $\mathsf{At}(X)$. The semantics for a propositional language is given by *interpretations* where an interpretation ω on At is a function $\omega : \mathsf{At} \to \{0, 1\}$ (where 0 stands for false and 1 stands for true). Let $\Omega(\mathsf{At})$ denote the set of all interpretations for At. An interpretation ω *satisfies* (or is a *model* of) an atom $a \in \mathsf{At}$, denoted by $\omega \models a$, if and only if $\omega(a) = 1$. The satisfaction relation $\models$ is extended to formulas in the usual way. For $\Phi \subseteq \mathcal{L}(\mathsf{At})$ we also define $\omega \models \Phi$ if and only if $\omega \models \phi$ for every $\phi \in \Phi$. Furthermore, for every set of formulas X, the set of models is $\mathsf{Mod}(X) = \{\omega \in \Omega(\mathsf{At}) \mid \omega \models X\}$. Define $X \models Y$ for (sets of) formulas X and Y if $\omega \models X$ implies $\omega \models Y$ for all ω.

Let $\top$ denote any tautology and $\perp$ any contradiction. If $\mathsf{Mod}(X) = \emptyset$ we write $X \models \perp$ and say that X is *inconsistent*.

2.1. Inconsistency Measurement

Inconsistency as defined above is a binary concept. To provide more fine-grained insights on inconsistency beyond such a binary classification, the field of inconsistency measurement [7] has evolved. The main objects of study in this field are *inconsistency measures*, which are quantitative measures that assess the degree of inconsistency for a knowledge base $\mathcal{K}$ with a non-negative numerical value. Intuitively, a higher value reflects a higher degree, or severity, of inconsistency. This can be useful for determining if one set of formulas is "more" inconsistent than another. Let $\mathbb{R}^{\infty}_{\geq 0}$ be the set of non-negative real values including ∞. Then, an inconsistency measure is defined as follows.

Definition 2. *An inconsistency measure $\mathcal{I}$ is any function $\mathcal{I} : \mathbb{K} \to \mathbb{R}^{\infty}_{\geq 0}$.*

To constrain the desired behavior of concrete inconsistency measures, several properties, called *rationality postulates*, have been proposed. A well-agreed upon property is that of *consistency*, which states that an inconsistency measure should return a value of 0 iff there is no inconsistency.

***Consistency* (CO)** $\mathcal{I}(\mathcal{K}) = 0$ if and only if $\mathcal{K}$ is consistent.

In the following, we consider only inconsistency measures that satisfy CO. But even CO, by itself, is too weak to characterize functions that intuitively measure inconsistency. Further important postulates introduced in [10] are *monotony, dominance* and *free-formula independence*, which we will define below. For that, we need some further notation.

First, a set $M \subseteq \mathcal{K}$ is called a *minimal inconsistent subset* (MIS) of $\mathcal{K}$ if $M \models\perp$ and there is no $M' \subset M$ with $M' \models\perp$. Let $\mathsf{MI}(\mathcal{K})$ be the set of all MISs of $\mathcal{K}$. Second, a formula $\alpha \in \mathcal{K}$ is called a *free formula* if $\alpha \notin \bigcup \mathsf{MI}(\mathcal{K})$. Let $\mathsf{Free}(\mathcal{K})$ be the set of all free formulas of $\mathcal{K}$. Similarly, we call $\alpha \in \mathcal{K}$ *problematic* if $\alpha \in \bigcup \mathsf{MI}(\mathcal{K})$, and denote $\mathsf{Problematic}(\mathcal{K})$ as the set of all problematic formulas. For example, if $\mathcal{K} = \{a, b, \neg b, c, \neg c, d \vee e\}$ then $\mathsf{MI}(\mathcal{K}) = \{\{b, \neg b\}, \{c, \neg c\}\}$, $\mathsf{Free}(\mathcal{K}) = \{a, d \vee e\}$, and $\mathsf{Problematic}(\mathcal{K}) = \{b, \neg b, c, \neg c\}$.

For the remainder of this section, let $\mathcal{I}$ be an inconsistency measure, $\mathcal{K}, \mathcal{K}' \in \mathbb{K}$, and $\alpha, \beta \in \mathcal{L}(\mathsf{At})$. Then, the basic postulates from [10] are defined as follows.

Monotony **(MO)** If $\mathcal{K} \subseteq \mathcal{K}'$ then $\mathcal{I}(\mathcal{K}) \leq \mathcal{I}(\mathcal{K}')$.

Free-formula independence **(IN)** If $\alpha \in \mathsf{Free}(\mathcal{K})$ then
$\mathcal{I}(\mathcal{K}) = \mathcal{I}(\mathcal{K} \setminus \{\alpha\})$.

Dominance **(DO)** If $\alpha \not\models\perp$ and $\alpha \models \beta$ then $\mathcal{I}(\mathcal{K} \cup \{\alpha\}) \geq \mathcal{I}(\mathcal{K} \cup \{\beta\})$.

MO states that adding formulas to the knowledge base cannot decrease the inconsistency value. IN means that removing free formulas from the knowledge base does not change the inconsistency value. DO consists of several cases, depending on the presence or absence of α or β in $\mathcal{K}$: the idea is that substituting a consistent formula α by a weaker formula β cannot increase the inconsistency.

Numerous inconsistency measures have been proposed (see [11] for a survey), many of which differ in regard to their compliance w.r.t. the introduced postulates. In this work, we will consider six measures as defined below. In order to define the *contension measure* $\mathcal{I}_c$ [12] we need some additional background on Priest's three-valued semantics [13]. A three-valued interpretation is a function $\nu : \mathsf{At} \to \{0, 1, \mathrm{B}\}$, which assigns to every atom either 0, 1 or B, where 0 and 1 correspond to $false$ and *true*, respectively, and B (standing for *both*) denotes a conflict. Assuming the *truth order* $\prec_T$ with $0 \prec_T B \prec_T 1$, the function ν can be extended to arbitrary formulas as follows: $\nu(\alpha \wedge \beta) = \min_{\prec_T}(\nu(\alpha), \nu(\beta))$, $\nu(\alpha \vee \beta) = \max_{\prec_T}(\nu(\alpha), \nu(\beta))$, $\nu(\neg\alpha) = 1$ if $\nu(\alpha) = 0$, $\nu(\neg\alpha) = 0$ if $\nu(\alpha) = 1$, and $\nu(\neg\alpha) = B$ if $\nu(\alpha) = B$. We say that an interpretation ν satisfies a formula α, denoted by $\nu \models^3 \alpha$, iff $\nu(\alpha) = 1$ or $\nu(\alpha) = \mathrm{B}$.

We will now define the measures used in this work.

Definition 3. *Let the measures $\mathcal{I}_d$, $\mathcal{I}_{\mathsf{MI}}$, $\mathcal{I}_p$, $\mathcal{I}_r$, $\mathcal{I}_c$, and $\mathcal{I}_{at}$ be defined as follows:*

$$\mathcal{I}_d(\mathcal{K}) = \begin{cases} 1 & \textit{if } \mathcal{K} \models \perp \\ 0 & \textit{otherwise} \end{cases}$$

$$\mathcal{I}_{\mathsf{MI}}(\mathcal{K}) = |\mathsf{MI}(\mathcal{K})|$$

$$\mathcal{I}_p(\mathcal{K}) = |\mathsf{Problematic}(\mathcal{K})|$$

$$\mathcal{I}_r(\mathcal{K}) = \min\{|X| \mid X \subseteq \mathcal{K} \textit{ and } \mathcal{K} \setminus X \not\models \perp\}$$

$$\mathcal{I}_c(\mathcal{K}) = \min\{|\nu^{-1}(B) \cap \mathsf{At}| \mid \nu \models^3 \mathcal{K}\}$$

$$\mathcal{I}_{at}(\mathcal{K}) = |\bigcup_{M \in \mathsf{MI}(\mathcal{K})} \mathsf{At}(M)|$$

A baseline approach is the drastic inconsistency measure $\mathcal{I}_d$ [14], which only differentiates between inconsistent and consistent knowledge bases. The MI-inconsistency measure $\mathcal{I}_{\mathsf{MI}}$ [14] counts the number of minimal inconsistent subsets. A similar version is the problematic inconsistency measure $\mathcal{I}_p$ [12], which counts the number of distinct formulas appearing in any inconsistent subset. The repair measure $\mathcal{I}_r$ counts the smallest number of formulas that must be removed in order to restore consistency. The contension measure $\mathcal{I}_c$ [12] quantifies inconsistency by seeking a three-valued interpretation that assigns B to a minimal number of propositions. We will formally define three-valued interpretations for LTL$_{\text{ff}}$ in the next subsection. The difference is that the propositional case is much simpler because there is no issue about states. Finally, the $\mathcal{I}_{at}$ measure counts the number of atoms in the non-free formulas.

We conclude this section with a small example illustrating the behavior of the considered inconsistency measures.

Example 1. *Consider $\mathcal{K}_3$, defined via*

$$\mathcal{K}_3 = \{a, \neg a, b, \neg b \wedge c \wedge d, \neg a \vee \neg b\}$$

Then we have that

$$\mathsf{MI}(\mathcal{K}_3) = \{\{a, \neg a\}, \{b, \neg b \wedge c \wedge d\}, \{a, \neg a \vee \neg b, b\}\}$$

Thus

$$\begin{array}{lll}\mathcal{I}_d(\mathcal{K}_3) = 1 & \mathcal{I}_{\mathsf{MI}}(\mathcal{K}_3) = 3 & \mathcal{I}_p(\mathcal{K}_3) = 5 \\ \mathcal{I}_r(\mathcal{K}_3) = 2 & \mathcal{I}_c(\mathcal{K}_3) = 2 & \mathcal{I}_{at}(\mathcal{K}_3) = 4\end{array}$$

The main focus of study in inconsistency measurement, and the introduced measures, has been on propositional logic. In this work, our aim is to apply inconsistency measures for linear time logic, which we introduce now.

2.2. Linear Temporal Logic on Fixed Traces

In this work, we consider a specific variant of $\mathrm{LTL_f}$ that we coin linear temporal logic on *fixed* traces ($\mathrm{LTL_{ff}}$). We consider a linear sequence of states $t_0, \ldots, t_m$, where every t_i is the state at instant i. We assume that $m > 1$ to avoid the trivial case. Note that the difference with $\mathrm{LTL_f}$—where interpretations can vary in their length as long as they are finite—is that we keep the length of this sequence finite and fixed across all interpretations. This variant of $\mathrm{LTL_f}$ is introduced mainly to discuss matters of inconsistency measurement, as here, the inconsistency value is computed in regard to a comparable length for all formulas. For example, if no fixed bound is assumed, and two sets of formulas over some finite traces of length n_1, n_2 are assessed to affect n_1 states, and n_2 states, respectively, it is not possible to ensure that this is due to the nature of the operators (such as $\mathbf{G}$), i. e., what we call "time-sensitive" inconsistency measurement, see Section 3. In practice, the length of traces can be naturally bounded by some (very) large integer, since all processes in a real company must be finished in some finite amount of time. However, the ideas presented in the next sections can be extended to $\mathrm{LTL_f}$ [15] in a straightforward manner by considering a sequence of increasing values for m and the limit. Furthermore, we will show below that a minimal length m for a set of formulas can be computed by considering the so-called *depth* of the formulas. We will discuss this further in Section 3.2. Note that the move to LTL is not as immediate, as some formulas might be sensitive to infinity [16]. We do not deal with LTL over infinite traces in this work.

The syntax of $\mathrm{LTL_{ff}}$ is the same as the syntax of LTL and $\mathrm{LTL_f}$ [17]. Formulas are built from a set of propositional symbols At and are closed under the Boolean connectives, the unary operator $\mathbf{X}$ (*next*), and the binary operator $\mathbf{U}$ (*until*). Formally, any formula φ of $\mathrm{LTL_{ff}}$ is built using the grammar rule

$$\varphi ::= a|(\neg\varphi)|(\varphi_1 \wedge \varphi_2)|(\varphi_1 \vee \varphi_2)|(\mathbf{X}\varphi)|(\varphi_1\mathbf{U}\varphi_2).$$

with $a \in \mathsf{At}$. Intuitively, $\mathbf{X}\varphi$ denotes that φ will hold at the next state and $(\varphi_1\mathbf{U}\varphi_2)$ denotes that φ_1 will hold until the state when φ_2 holds. Let $d(\varphi) \in \mathbb{N}$ denote the maximal number of nested temporal operators in φ.[3]

From the basic operators, some useful abbreviations can be derived, including $\mathbf{F}\varphi$ (defined as $\top\mathbf{U}\varphi$), which denotes that φ will hold eventually over the linear sequence (possibly including the current state) and $\mathbf{G}\varphi$ (defined as $\neg\mathbf{F}\neg\varphi$), which denotes that φ will hold for all (following) states including the current one. Here $\top$ is any tautology and $\bot$ is any contradiction.

An LTL$_{\mathrm{ff}}$-interpretation $\hat{\omega}$ w.r.t. At is a function mapping each state and proposition to 0 or 1, meaning that $\hat{\omega}(t, a) = 1$ if proposition a is assigned 1 (true) in state t.[4] Then the satisfaction of a formula ϕ by an interpretation $\hat{\omega}$, denoted by $\hat{\omega} \models \phi$, is defined via

$$\hat{\omega} \models \phi \quad \Leftrightarrow \quad \hat{\omega}, t_0 \models \phi$$

where $\hat{\omega}, t_i \models \phi$ for any interpretation $\hat{\omega}$ as above and for every $t_i \in \{t_0, \ldots, t_m\}$ is inductively defined as follows:

$$\begin{aligned}
&\hat{\omega}, t_i \models a \text{ iff } \hat{\omega}(t_i, a) = 1 \text{ for } a \in \mathsf{At}\\
&\hat{\omega}, t_i \models \neg\varphi \text{ iff } \hat{\omega}, t_i \not\models \varphi\\
&\hat{\omega}, t_i \models \varphi_1 \wedge \varphi_2 \text{ iff } \hat{\omega}, t_i \models \varphi_1 \text{ and } \hat{\omega}, t_i \models \varphi_2\\
&\hat{\omega}, t_i \models \varphi_1 \vee \varphi_2 \text{ iff } \hat{\omega}, t_i \models \varphi_1 \text{ or } \hat{\omega}, t_i \models \varphi_2\\
&\hat{\omega}, t_i \models \mathbf{X}\varphi \text{ iff } i < m \text{ and } \hat{\omega}, t_{i+1} \models \varphi\\
&\hat{\omega}, t_i \models \varphi_1\mathbf{U}\varphi_2 \text{ iff either } \hat{\omega}, t_i \models \varphi_1 \wedge \varphi_2 \text{ or}\\
&\qquad (\hat{\omega}, t_j \models \varphi_2 \text{ for some } j \in \{i+1, \ldots, m\}\\
&\qquad \text{and } \hat{\omega}, t_k \models \varphi_1 \text{ for all } k \in \{i, \ldots, j-1\})
\end{aligned}$$

An interpretation $\hat{\omega}$ satisfies a set of formulas K iff $\hat{\omega} \models \phi$ for all $\phi \in K$. A set K is consistent iff there exists $\hat{\omega}$ such that $\hat{\omega} \models K$ (Otherwise, we say

[3]$d(\varphi)$ is inductively defined via $d(a) = 0$ for $a \in \mathsf{At}$, $d(\neg\phi) = d(\phi)$, $d(\phi_1 \wedge \phi_2) = d(\phi_1 \vee \phi_2) = \max\{d(\phi_1), d(\phi_2)\}$, $d(\mathbf{X}\phi) = 1 + d(\phi)$, and $d(\phi_1\mathbf{U}\phi_2) = 1 + \max\{d(\phi_1), d(\phi_2)\}$.

[4]Recall that we assume time of a fixed length $t_0, \ldots, t_m$ and interpretations only vary in what is true at each state.

K is inconsistent). Define $X \models Y$ for (sets of) formulas X and Y if $\hat{\omega} \models X$ implies $\hat{\omega} \models Y$ for all $\hat{\omega}$.

2.3. Related Work and Contributions

The topic of the present work is related to several fields and aspects of *inconsistency diagnosis in LTL, inconsistency measurement, defeasible reasoning*, and *answer set programming*, that we discuss in the following.

Inconsistency Diagnosis in LTL. The core focus of this work is related to consistency and model checking in declarative process specifications, see e. g. [18, 19, 20]. For the general task of diagnosis of LTL-based specifications, there have been some works that allow to diagnose satisfiability, or compute inconsistent cores (e. g., sets of formulas) [3, 21, 4, 22]. In this regard, our approach extends recent works [4, 5, 3, 20] on the identification of inconsistent sets in declarative process specifications by allowing to look "into" those sets and leverage inconsistency resolution with quantitative insights. For example, existing resolution approaches mainly try to minimize the *number* of deleted formulas [4, 5, 23]. This, however, completely leaves aside the semantics of those formulas or their impact on any corresponding process, in particular, how many points in time are affected by a formula. Given this motivation, it is useful to consider also the degree to which certain formulas affect the following behavior. In this work, we coin such an analysis as *time sensitive* inconsistency measurement.

Approaches for Handling Inconsistent Information. While the field of inconsistency measurement has brought forward various approaches for handling inconsistency in propositional logic formalisms, results on measuring inconsistency in logics with modal operators such as time are still rare. This paper is related to [24] which presents several, what we call time sensitive, inconsistency measures for branching time logics (BTL). However, in this work we are able to avoid the complicated overload of branching time as the process specifications are provided in linear time logic. Using branching time logic adds a layer of complexity that is unnecessary when dealing with a linear time situation. Just to take one example, consider the set $\{\mathbf{X}a, \mathbf{X}\neg a\}$. In linear time logic, this gives one conflict at the next state. But in the case of branching time logic, this highly depends on additional quantification operators. If only considering "some next state", then the set is consistent because a and $\neg a$ may hold in different next states. If considering "all next states", then it is inconsistent but how inconsistent depends on the number of next states. We avoid such issues by dealing only with linear temporal logic. Note also that BTL takes a different view on time than $\mathrm{LTL_f}$ as studied in this paper and is therefore expressively incomparable (cf. [25]).

As a framework for measuring inconsistency in declarative specifications, we develop a paraconsistent semantics. Paraconsistent reasoning [13] in general can be useful for reasoning about knowledge bases in a meaningful way even in the presence of contradictions. To this extent, we lift the existing results on paraconsistent semantics for propositional logics (and the corresponding approaches for measuring inconsistency based on these semantics) to the setting of LTL. Paraconsistent semantics for LTL has already been studied, as shown in [26] and [27]. The emphasis in those works is to develop a sound and complete proof theory for several paraconsistent versions of LTL. Both the bounded and unbounded versions of LTL are considered. Connections are shown with intuitionistic logic as well as a paraconsistent 4-valued logic. For our paper we use the simplest 3-valued paraconsistent logic that serves our purpose in measuring inconsistency.

Defeasible Reasoning. The approach of paraconsistent reasoning, or inconsistency-tolerant reasoning, is also connected to *defeasible reasoning* [28]. In defeasible logic, beliefs can be overruled by others, e. g., to model explicit exceptions (in this sense, different formulas of the knowledge base can be said to have a different priority). As the ability to model exceptions may be important in settings of declarative process specifications, we extend the paraconsistent semantics to be able to consider *preference relations* between formulas. In this way, modellers have the ability to express default rules and exceptions, e. g, *"generally $\neg\phi$ should hold in all states, but in the first state, ϕ may hold".* To the best of our knowledge, results on preference relations in declarative process specification are still very underdeveloped, which is why we believe this to be a valuable extension to current research, e. g., modelling exceptions in Declare specifications by means of modelling priorities. This could be very valuable in settings where data is also considered (e. g. "generally some rule should be true unless certain data attributes such as resources are observed"), or in settings where there might be some notion of "uncertainty" in regard to the specific formulas of a specification. In [29], a semantics for defeasible LTL has been proposed; however, those authors propose novel modal operators such as "defeasibly $\mathbf{X}$". This means that such an approach is not directly applicable for settings of declarative process specifications, e. g., for Declare specifications, as the modeller would need to redesign the Declare language to be built over these defeasible operators. Instead, in this work, we propose to add preference relations, which offers a decoupled and less obstrusive way of facilitating defeasible reasoning in declarative process specifications. Also, in [30], the authors discuss a variant of defeasible logic with temporal operators, however, the rule formalism there is very different in which all rules have a fixed start and end point, which is not the case in our setting, e. g., due to the use of implications.

A work which can be considered slightly related here is that on probabilistic Declare [31, 32], i. e., for declarative constraints over uncertain information. However, while this allows for capturing the quantitative priorisation, it does not allow for modeling explicit relations such as exceptions. This, however, is possible in our work, as we introduce an explicit preference relation between formulas.

For reasoning about specification in the presence of preference relations, we develop a new semantics. For example, if we consider $\{\mathbf{G}\neg a, \mathbf{X}a\}$ and assume $\mathbf{X}a$ overrules $\mathbf{G}\neg a$, then this specification should be viewed as consistent ($\mathbf{X}a$ is seen as an exception to $\mathbf{G}\neg a$). To enable such semantics, we utilize the concept of *taggings* as suggested by [28, 33]. In essence, a tagging is a meta-statement about an element of a knowledge base, stating whether this element should or should not hold. Note however, that in [28, 33], the formalism considered is of a logic program form, meaning that all the rules consist only of literals in conjunctive form. This enables those authors to define taggings for individual literals. In our setting, this is not sufficient, as the literals can be connected via arbitrary boolean connectors, and we include modal operators. Therefore, we will present a new tagging system which allows the assignment of tags to *formulas* of the knowledge base (which in turn allows for inferring which formulas should hold, and which may be overruled by others).

Note also that inconsistency measurement in settings where individual formulas can have different priorities has also been addressed in the context of stratified knowledge bases [34]. However, in [34], all elements can be ordered into different strata (or layers) of a knowledge base, such that all elements from a higher layer are seen as more important than those of lower layers. In our setting, this is different, as we only have a partial ordering based on individual preferences. Also, a distinction of our approach (defeasible logic) to stratified knowledge bases is that in our approach, default rules and their exceptions can be defined, which is different from a stratified approach, where formulas are ordered by priority, but no exception relations can me modelled individually.

Answer Set Programming. To implement our results, we develop encodings of the proposed LTL semantics in answer set programming (ASP).[5] We adapt this approach as this declarative programming paradigm has proven itself for the task of computing (paraconsistent) interpretations (in propositional logic) [35], which is our basis for measuring inconsistency. In this work, we present a novel encoding to handle the temporal operators of LTL. A related work, [36], also presents an ASP-based encoding for LTL_f. However, those authors define the semantics of LTL_f via finite state automata, which also means that, for encoding formulas, a transformation of the formulas into an automaton representation is required in advance. As such operations introduce a computational burden [4], in this work, we aim to implement the semantics directly within ASP. Next to the computational aspect, a further distinction is that the approach in [36] is "optimized" for the Declare modeling language, meaning that only a predefined set of Declare constraint types is supported. In our approach, as we implement the semantics of LTL directly; this allows to support arbitrary formulas, which is currently not possible out-of-the-box in [36].

Another related work is by Heljanko and Niemelä [37], who proposed an ASP approach for bounded model checking in LTL. Similarly to our approach, the authors encode LTL semantics directly. However, in order to tackle the problem of inconsistency measurement, we need to encode a different, three-valued semantics for LTL, which requires more intricate encodings, in particular for the **U** operator.

3. A Framework for Inconsistency Measurement in LTL_ff

In this section, we introduce a framework for measuring inconsistency in LTL_ff. As we will show, existing inconsistency measures cannot provide meaningful insights when dealing with temporal logic. Therefore, we develop a novel paraconsistent semantics as a framework for handling inconsistency in LTL_ff.

[5]An additional introduction to ASP can be found in Appendix B.

3.1. Motivation for Inconsistency Measures for LTL$_{\mathit{ff}}$

We recall the sets of LTL$_{\mathrm{ff}}$ formulas $\mathcal{K}_1$ and $\mathcal{K}_2$:

$$\mathcal{K}_1 = \{\mathbf{X}a, \mathbf{X}\neg a\} \qquad\qquad \mathcal{K}_2 = \{\mathbf{G}a, \mathbf{G}\neg a\}$$

The knowledge base $\mathcal{K}_1$ states that a is both true and false in the next state while $\mathcal{K}_2$ states that a is both true and false in all future states. Obviously, both knowledge bases are inconsistent. Yet, the inconsistencies are different in regard to the number of states they affect. For $\mathcal{K}_1$ the number is 1 and for $\mathcal{K}_2$ the number is $m > 1$. It would therefore be desirable for an inconsistency measure to take this information into account and assign $\mathcal{K}_2$ a larger inconsistency value.

In order to capture LTL$_{\mathrm{ff}}$ by the inconsistency measurement framework of Section 2.1, from now on a knowledge base $\mathcal{K}$ (Definition 1) will be a finite set of LTL$_{\mathrm{ff}}$ formulas and $\mathbb{K}$ is the set of all LTL$_{\mathrm{ff}}$ knowledge bases. So we can apply the introduced inconsistency measures for $\mathcal{K}_1$ and $\mathcal{K}_2$ in a straightforward manner.

Example 2. *Consider $\mathcal{K}_1$ and $\mathcal{K}_2$. Then we have that*

$$\begin{aligned} \mathcal{I}_d(\mathcal{K}_1) &= 1 & \qquad \mathcal{I}_d(\mathcal{K}_2) &= 1 \\ \mathcal{I}_{\mathsf{MI}}(\mathcal{K}_1) &= 1 & \mathcal{I}_{\mathsf{MI}}(\mathcal{K}_2) &= 1 \\ \mathcal{I}_p(\mathcal{K}_1) &= 2 & \mathcal{I}_p(\mathcal{K}_2) &= 2 \\ \mathcal{I}_r(\mathcal{K}_1) &= 1 & \mathcal{I}_r(\mathcal{K}_2) &= 1 \\ \mathcal{I}_c(\mathcal{K}_1) &= 1 & \mathcal{I}_c(\mathcal{K}_2) &= 1 \\ \mathcal{I}_{at}(\mathcal{K}_1) &= 1 & \mathcal{I}_{at}(\mathcal{K}_2) &= 1 \end{aligned}$$

Note that all six inconsistency measures give identical values for $\mathcal{K}_1$ and $\mathcal{K}_2$, because they, or for that matter, any other propositional logic inconsistency measure, cannot distinguish between $\mathbf{X}$ and $\mathbf{G}$. But intuitively $\mathcal{K}_2$ is more inconsistent than $\mathcal{K}_1$ because the inconsistency persists through all future states in $\mathcal{K}_2$ as opposed to the single state in $\mathcal{K}_1$. Thus, we believe that a proper inconsistency measure for LTL$_{\mathrm{ff}}$ should distinguish between these operators. Therefore, we propose a new rationality postulate.

Time Sensitivity **(TS)** For all formulas φ of propositional logic, $\mathcal{I}(\{\mathbf{G}\varphi, \mathbf{G}\neg\varphi\}) > \mathcal{I}(\{\mathbf{X}\varphi, \mathbf{X}\neg\varphi\})$.

In other words, the number of affected states should be reflected in the inconsistency value, i. e., inconsistency measures for LTL$_{\text{ff}}$ should be time sensitive.

Proposition 1. $\mathcal{I}_d, \mathcal{I}_{\mathsf{MI}}, \mathcal{I}_p, \mathcal{I}_r, \mathcal{I}_c, \mathcal{I}_{at}$ *violate* *TS.*

Following Proposition 1, the existing measures that we have from propositional logic cannot capture the desired behavior. Therefore, we introduce a novel approach to measure inconsistency in LTL$_{\text{ff}}$.

3.2. A Paraconsistent Semantics for LTL$_{ff}$

Our framework for measuring inconsistency in LTL$_{\text{ff}}$ is an LTL$_{\text{ff}}$-variant of the three-valued semantics of [13]. A three-valued interpretation $\hat{\nu}$ for LTL$_{\text{ff}}$ is a function mapping each state and proposition to 0, 1 or B, that is, $\hat{\nu} : \{t_0, t_1, \ldots t_m\} \times \mathsf{At} \rightarrow \{0, 1, \mathrm{B}\}$ where as before 0 and 1 correspond to the classic logical false and true, respectively, and B (standing for *both*) denotes a conflict. We then assign

$$\hat{\nu}(\phi) = \hat{\nu}(t_0, \phi)$$

where $\hat{\nu}(t_i, \phi)$, for any interpretation $\hat{\nu}$ as above and state $t_i \in \{t_0, \ldots, t_m\}$, is inductively defined as follows:

$$\hat{\nu}(t_i, a) = \hat{\nu}(t_i, a) \text{ for } a \in \mathsf{At}$$

$$\hat{\nu}(t_i, \neg\phi) = \begin{cases} 1 & \text{if } \hat{\nu}(t_i, \phi) = 0 \\ 0 & \text{if } \hat{\nu}(t_i, \phi) = 1 \\ \mathrm{B} & \text{if } \hat{\nu}(t_i, \phi) = \mathrm{B} \end{cases}$$

$$\hat{\nu}(t_i, \varphi_1 \wedge \varphi_2) = \begin{cases} 1 & \text{if } \hat{\nu}(t_i, \varphi_1) = \hat{\nu}(t_i, \varphi_2) = 1 \\ 0 & \text{if } \hat{\nu}(t_i, \varphi_1) = 0 \text{ or } \hat{\nu}(t_i, \varphi_2) = 0 \\ \mathrm{B} & \text{otherwise} \end{cases}$$

$$\hat{\nu}(t_i, \varphi_1 \vee \varphi_2) = \begin{cases} 1 & \text{if } \hat{\nu}(t_i, \varphi_1) = 1 \text{ or } \hat{\nu}(t_i, \varphi_2) = 1 \\ 0 & \text{if } \hat{\nu}(t_i, \varphi_1) = \hat{\nu}(t_i, \varphi_2) = 0 \\ \mathrm{B} & \text{otherwise} \end{cases}$$

$$\hat{\nu}(t_i, \mathbf{X}\varphi) = \begin{cases} \hat{\nu}(t_{i+1}, \varphi) & \text{if } i < m \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{\nu}(t_i, \varphi_1 \mathbf{U} \varphi_2) = \begin{cases} 1 & \text{if either } \hat{\nu}(t_i, \phi_1 \wedge \phi_2) = 1 \text{ or there is} \\ & j \in \{i+1, \ldots, m\} \text{ with } \hat{\nu}(t_j, \phi_2) = 1 \text{ and} \\ & \hat{\nu}(t_i, \phi_1) = \ldots \hat{\nu}(t_{j-1}, \phi_1) = 1, \qquad \text{otherwise:} \\ \mathrm{B} & \text{if either } \hat{\nu}(t_i, \phi_2) = B \text{ or there is} \\ & j \in \{i+1, \ldots, m\} \text{ with } \hat{\nu}(t_j, \phi_2) \in \{1, B\} \text{ and} \\ & \{\hat{\nu}(t_i, \phi_1), \ldots, \hat{\nu}(t_{j-1}, \phi_1)\} \subseteq \{1, B\} \\ 0 & \text{otherwise} \end{cases}$$

Some comments on the above definition are in order. First, note that the evaluation of the classical Boolean connectives is the same as for propositional three-valued semantics (see Section 2.1). Furthermore, the evaluation of $\mathbf{X}\phi$ is simply the truth value of ϕ at the next state, or, if there is no next state, 0 (as for the classical semantics of $\mathrm{LTL}_{\mathrm{ff}}$). The main new feature, however, is the three-valued evaluation of a formula of the form $\varphi_1 \mathbf{U} \varphi_2$. This formula evaluates to 1 if ϕ_2 evaluates to 1 in state t_i or in some future state and ϕ_1 evaluates to 1 in between. Otherwise, we evaluate $\varphi_1 \mathbf{U} \varphi_2$ to B if ϕ_2 evaluates to B in state t_i or to 1 or B in some future state and ϕ_1 evaluates to 1 or B in between (so at least one of these evaluations must be to B). If neither case holds, then $\varphi_1 \mathbf{U} \varphi_2$ evaluates to 0, i. e., if either ϕ_2 always evaluates to 0 in the future or in-between φ_1 evaluates at least once to 0.

A three-valued $\mathrm{LTL}_{\mathrm{ff}}$ interpretation $\hat{\nu}$ satisfies a formula ϕ, denoted by $\hat{\nu} \models^3 \phi$, iff $\hat{\nu}(t_0, \phi) \in \{1, \mathrm{B}\}$. A three-valued interpretation $\hat{\nu}$ satisfies a set of formulas $\mathcal{K}$ iff $\hat{\nu} \models^3 \phi$ for all $\phi \in \mathcal{K}$.

Example 3. *Let* $\mathsf{At} = \{a, b\}$ *and assume* $m = 2$. *Consider the knowledge base* $\mathcal{K}_4$, *defined via*

$$\mathcal{K}_4 = \{\boldsymbol{X}\neg a, a\,\boldsymbol{U}b\}$$

and the three-valued interpretation $\hat{\nu}$ *defined via*

$$\begin{aligned} \hat{\nu}(t_0, a) &= 1 & \qquad \hat{\nu}(t_0, b) &= 0 \\ \hat{\nu}(t_1, a) &= B & \qquad \hat{\nu}(t_1, b) &= 0 \\ \hat{\nu}(t_2, a) &= 0 & \qquad \hat{\nu}(t_2, b) &= 1 \end{aligned}$$

Then we have $\hat{\nu}(t_0, a\,\boldsymbol{U}b) = B$ *as* b *evaluates to* 1 *in* t_2 *and* a *evaluates to* B *in* t_1. *Moreover, we have* $\hat{\nu}(t_0, \boldsymbol{X}\neg a) = B$ *and therefore* $\hat{\nu} \models^3 \mathcal{K}$.

Define $X \models^3 Y$ for formulas X and Y if $\hat{\nu} \models X$ implies $\hat{\nu} \models Y$ for all $\hat{\nu}$.

Example 4. *Let* $\mathsf{At} = \{a, b, c\}$. *Consider the knowledge base* $\mathcal{K}_5$, *defined via*

$$\mathcal{K}_5 = \{\boldsymbol{G}\neg a, \boldsymbol{G}\neg b, a\,\boldsymbol{U}b, \boldsymbol{G}c\}$$

Note that $\mathcal{K}_5$ *is inconsistent under classical LTL*$_{ff}$ *semantics for every* $m > 1$ *(the first three formulas require* a *and* b *to be always false and that* a *has to hold until* b *holds), so we have* $\mathcal{K}_5 \models \phi$ *for every* ϕ *of LTL*$_{ff}$ *(in classical two-valued semantics, LTL*$_{ff}$ *conforms to the principle of explosion.). In particular, we have* $\mathcal{K}_5 \models \boldsymbol{G}c$ *and* $\mathcal{K}_5 \models \neg\boldsymbol{G}c$, *contrary to intuition since* $\boldsymbol{G}c$ *is explicitly part of* $\mathcal{K}_5$ *and "not involved" in the inconsistency in* $\mathcal{K}_5$. *Using our three-valued semantics, we obtain* $\mathcal{K}_5 \models^3 \boldsymbol{G}c$ *and* $\mathcal{K}_5 \not\models^3 \neg\boldsymbol{G}c$ *as desired.*

In the propositional logic case, $\models^3$ is a faithful extension of $\models$, meaning that $\omega \models \phi$ if and only if $\omega \models^3 \phi$ for every two-valued interpretation ω and every ϕ. Our LTL$_{ff}$ extension of the three-valued semantics enjoys the same property (note that every two-valued interpretation is also a three-valued interpretation that does not use the value B).

Proposition 2. *For every (two-valued) LTL*$_{ff}$ *interpretation* $\hat{\omega}$ *and LTL*$_{ff}$ *formula* ϕ, $\hat{\omega} \models \phi$ *if and only if* $\hat{\omega} \models^3 \phi$.

The three-valued semantics of [13] has another nice property in propositional logic, namely the non-existence of inconsistency: every propositional formula is trivially satisfiable by the interpretation that assigns B to all propositions. In general, an LTL$_{ff}$ formula may become unsatisfiable w.r.t. to the three-valued semantics if it affects a state "beyond" t_m. However, for other formulas we obtain the following result regarding satisfiability.

Proposition 3. *For any LTL$_{\mathrm{ff}}$ formula ϕ with $d(\phi) \leq m$ there is $\hat{\nu}$ with $\hat{\nu} \models^3 \phi$.*

The semantics presented in this section allows for inconsistency-tolerant reasoning in LTL$_{\mathrm{ff}}$ (and it can straightforwardly be adapted for LTL$_{\mathrm{f}}$ and LTL). In particular, via Proposition 3, m can be set to the depth of the conjunction of all formulas in a knowledge base $\mathcal{K}$. Then, this ensures an interpretation exists. It directly follows that if an inconsistency affects $n < m$ states, it is never possible that all states will be affected. In this case, if one wants to support loops, the concrete number of affected states would only grow constantly with the number of loops (so setting m to the depth suffices in this case). In Section 7.3, we will also show how–in case the inconsistency affects exactly m states–it can easily be approximated that this holds for any m, should m be increased.

4. Approaches for Time Sensitive Inconsistency Measurement in LTL$_{\mathrm{ff}}$

In the following, we present approaches for *measuring (overall) inconsistency, measuring inconsistency in the presence of preference relations*, and *measuring element-based inconsistency.*

4.1. Baseline Measures

We will now exploit our three-valued semantics for LTL$_{\mathrm{ff}}$ to define inconsistency measures. We do this similarly as for propositional logic by assessing the amount of usage of the paraconsistent truth value B in models of an LTL$_{\mathrm{ff}}$ knowledge base $\mathcal{K}$ but refine it by two different levels of granularity. This yields two new inconsistency measures.

Our first approach measures the number of states affected by inconsistency. For any three-valued interpretation $\hat{\nu}$, define

$$\mathsf{AffectedStates}(\hat{\nu}) = \{t \mid \exists a : \hat{\nu}(t, a) = \mathrm{B}\}$$

In other words, $\mathsf{AffectedStates}(\hat{\nu})$ is the set of states where $\hat{\nu}$ assigns B to at least one proposition. We can define an inconsistency measure by considering those 3-valued models of the knowledge base that affect the minimal number of states.

Definition 4 (LTL time measure)**.** *Let $\mathcal{K}$ be a set of formulas. Then, the LTL time measure is defined via*

$$\mathcal{I}_d^{LTL}(\mathcal{K}) = \min_{\hat{\nu} \models^3 \mathcal{K}} |\mathit{AffectedStates}(\hat{\nu})|$$

if there is $\hat{\nu}$ with $\hat{\nu} \models^3 \mathcal{K}$ and $\mathcal{I}_d^{LTL}(\mathcal{K}) = \infty$ otherwise.

This measure counts the number of states for which the knowledge base is inconsistent. It is, in fact, the extension of the drastic measure, $\mathcal{I}_d$, in that for each state it adds 1 if there is an inconsistency and 0 otherwise. This measure can be used to distinguish the knowledge bases $\mathcal{K}_1$ and $\mathcal{K}_2$, i. e., it is time sensitive.

Example 5. *We recall the knowledge bases $\mathcal{K}_1 = \{\boldsymbol{X}a, \boldsymbol{X}\neg a\}$ and $\mathcal{K}_2 = \{\boldsymbol{G}a, \boldsymbol{G}\neg a\}$. Then we have*

$$\mathcal{I}_d^{LTL}(\mathcal{K}_1) = 1 \qquad\qquad \mathcal{I}_d^{LTL}(\mathcal{K}_2) = m$$

As an example where there is no $\hat{\nu}$ s.t. $\hat{\nu} \models^3 \mathcal{K}$, consider the formula $\mathbf{XXX}a$. This formula cannot be satisfied for $m = 2$, so $\mathcal{I}_d^{LTL}$ would return ∞ here.

Example 5 shows that the proposed measure $\mathcal{I}_d^{LTL}$ can already provide meaningful (i. e., time-sensitive) insights for measuring inconsistency in LTL. But a potential limitation is that it can only distinguish inconsistency in individual states in a binary manner. For example, $\mathcal{I}_d^{LTL}$ cannot distinguish the knowledge base $\mathcal{K}_6 = \{\mathbf{X}a, \mathbf{X}\neg a, \mathbf{X}b, \mathbf{X}\neg b\}$ from $\mathcal{K}_1$ because all inconsistencies occur at one state, namely t_1. For this reason we believe it is useful to be able to look inside states for inconsistency. In order to do so, given a three-valued interpretation $\hat{\nu}$, define

$$\mathsf{ConflictBase}(\hat{\nu}) = \{(t, a) \mid \hat{\nu}(t, a) = \mathrm{B}\}$$

Then, define the LTL contension measure as follows.

Definition 5 (LTL contension measure)**.** *Let $\mathcal{K}$ be a set of formulas and*

$$\mathcal{I}_c^{LTL}(\mathcal{K}) = \min_{\hat{\nu} \models^3 \mathcal{K}} |\mathit{ConflictBase}(\hat{\nu})|$$

if there is $\hat{\nu}$ with $\hat{\nu} \models^3 \mathcal{K}$ and $\mathcal{I}_c^{LTL}(\mathcal{K}) = \infty$ otherwise.

$\mathcal{I}_c^{LTL}$ seeks an interpretation that assigns B to a minimal number of propositions individually over all the states and uses this number for the inconsistency measure. This is an extension of $\mathcal{I}_d^{LTL}$, and for that matter, of $\mathcal{I}_c$ as it calculates $\mathcal{I}_c^{LTL}$ for each state and sums the numbers obtained this way.

Example 6. *We recall the knowledge bases* $\mathcal{K}_1 = \{\boldsymbol{X}a, \boldsymbol{X}\neg a\}$, $\mathcal{K}_6 = \{\boldsymbol{X}a, \boldsymbol{X}\neg a, \boldsymbol{X}b, \boldsymbol{X}\neg b\}$*, and consider* $\mathcal{K}_7 = \{\boldsymbol{G}a, \boldsymbol{G}\neg a, \boldsymbol{G}b, \boldsymbol{G}\neg b\}$. *If* $m = 3$*, then we have*

$$\begin{array}{lll} \mathcal{I}_d^{LTL}(\mathcal{K}_1) = 1 & \mathcal{I}_d^{LTL}(\mathcal{K}_6) = 1 & \mathcal{I}_d^{LTL}(\mathcal{K}_7) = 3 \\ \mathcal{I}_c^{LTL}(\mathcal{K}_1) = 1 & \mathcal{I}_c^{LTL}(\mathcal{K}_6) = 2 & \mathcal{I}_c^{LTL}(\mathcal{K}_7) = 6 \end{array}$$

As can be seen in Example 6, the two inconsistency measures proposed in this work can, contrary to previously existing measures, be used to provide meaningful insights into inconsistency in linear temporal logic, i. e., they are in fact time sensitive. As the two measures have a different granularity in regard to time, selecting which of the two to use depends on the intended use case. More specifically, the proposed measures have a different *expressivity* [38] with respect to the number of states. That is, $\mathcal{I}_d^{LTL}$ and $\mathcal{I}_c^{LTL}$ have a different number of inconsistency values they can get for a given number of states, as we now show.

Definition 6. *Let* $\mathcal{I}$ *be an inconsistency measure. Then, define the expressivity of* $\mathcal{I}$ *for* m *as* $\mathcal{C}(\mathcal{I}, m) = |\{\mathcal{I}(\mathcal{K}) \mid \mathcal{K} \in \mathbb{K}\}|$*, for a sequence of states* $t_0, ..., t_m$.

In other words, $\mathcal{C}(\mathcal{I}, m)$ denotes the number of different inconsistency values $\mathcal{I}$ assigns to knowledge bases, assuming (at most) a sequence of states $t_0, ..., t_m$. For any interpretation, we have that any number of states $t_0, \ldots, t_m$, or no states at all can be affected. This naturally limits the maximum number of AffectedStates. The size of the ConflictBase can grow to infinity (due to the possibility of arbitrarily large signatures), as all conflicts for each state are considered.

Proposition 4. $\mathcal{C}(\mathcal{I}_d^{LTL}, m) = m + 2$ *and* $\mathcal{C}(\mathcal{I}_c^{LTL}, m) = \infty$.

Intuitively, it would be possible to devise further time-sensitive inconsistency measures for LTL_{ff}. We will however leave this discussion for future work. Importantly, the aim of this paper is to show that traditional inconsistency measures cannot be plausibly applied to temporal logics, and to present means for time sensitive inconsistency measurement. In this regard,

$\mathcal{I}$	CO	MO	IN	DO	TS
$\mathcal{I}_d$	✓	✓	✓	✓	✗
$\mathcal{I}_{\mathsf{MI}}$	✓	✓	✓	✗	✗
$\mathcal{I}_p$	✓	✓	✓	✗	✗
$\mathcal{I}_r$	✓	✓	✓	✗	✗
$\mathcal{I}_c$	✓	✓	✗	✓	✗
$\mathcal{I}_{at}$	✓	✗	✗	✗	✗
$\mathcal{I}_d^{LTL}$	✓	✓	✓	✓	✓
$\mathcal{I}_c^{LTL}$	✓	✓	✗	✓	✓

Table 1: Compliance of inconsistency measures with rationality postulates.

the measures proposed in this work can be used as a baseline for measuring inconsistency in LTL. Also, they (broadly) satisfy other desirable properties and can therefore be seen as strictly better (w.r.t. the considered postulates) than their propositional logic "counterpart", i. e., $\mathcal{I}_d$ for $\mathcal{I}_d^{LTL}$, respectively $\mathcal{I}_c$ for $\mathcal{I}_c^{LTL}$. The results of this section are summarized in Table 1.

Proposition 5. *The compliance of the inconsistency measures $\mathcal{I}_d$, $\mathcal{I}_{\mathsf{MI}}$, $\mathcal{I}_p$, $\mathcal{I}_r$, $\mathcal{I}_c$, $\mathcal{I}_{at}$, $\mathcal{I}_d^{LTL}$ and $\mathcal{I}_c^{LTL}$ with the postulates CO, MO, IN, DO and TS is as shown in Table 1.*

Note that only the measures we introduced satisfy TS. Note also that $\mathcal{I}_c^{LTL}$ does not satisfy IN due to the problem of iceberg inconsistencies, explained in the proof given in the Appendix.

4.2. Measuring Inconsistency in LTL$_{ff}$ with Preferences

So far, we have considered LTL knowledge bases consisting of a set of (equally valid) formulas. In this sense, the considered formalism is monotonic, i. e., for two sets of formulas X, Y over At, if $X \models \phi$ then we have that $X \cup Y \models \phi$. However, in practical settings, this form of monotonicity may be too strict. For example, a modeller of a process specification may need to specify exceptions. Take for example the formula $\mathbf{G}\neg a$, which states that the task a should never occur at any point in time. In a monotonic setting, it is not possible to refute this. However, it may be necessary to model the exception that a should still be allowed to occur in the first time point. Importantly, if one models such an exception, the formulas of the knowledge base should not be viewed as inconsistent, as the specification that (1) a should generally never occur, but (2) as an exception, a is allowed to occur in the first instant is explicitly desired. Intuitively, (2) overrules (1). So preference relations are closely connected to the inconsistency of the specification.

There are many use-cases in which modellers may want to decide for modelling preferences between formulas, instead of rewriting (multiple) formulas to incorporate the exception – many of which are related to belief revision or dynamic epistemic behavior in general. First, knowledge might be modelled over time and can evolve, so having the ability to model preferences can be helpful to model exceptions in a flexible manner without the need to rework the original formulas. Furthermore, there might be settings where knowledge comes from different sources or considers different data perspectives. Here, being able to model explicit relations between different pieces of knowledge can help in the human comprehension of the different aspects that need to be considered. We will provide further examples on how preference relations can be useful for Declare in Section 5.

In general, the notion of a preference relation is strongly connected to defeasible reasoning [28], where beliefs can be modelled as defeasible, meaning that they can be "overruled" by other beliefs, as illustrated in the example. To incorporate this aspect in our framework, we propose the following extension including *preference relations*.

Definition 7 (Knowledge base with a preference relation)**.** *Let $\mathcal{K}$ be a knowledge base over* At *as before. A knowledge base with a preference relation $\mathcal{K}_>$ is a pair $\mathcal{K}_> = (\mathcal{K}, >)$, where $> \subseteq \mathcal{K} \times \mathcal{K}$ is a preference relation between elements in $\mathcal{K}$. Let $\mathbb{K}_>$ be the set of all such knowledge bases.*

For readability, we still refer to such knowledge bases with a preference relation simply as "knowledge bases" and mark them as $\mathcal{K}_>$. If $(\phi_1, \phi_2) \in >$, we say that ϕ_1 is preferred to ϕ_2 (see also below for semantics). For readability, we add an identifier f_i : to every formula, and denote a preference $(f_1 : \phi_1, f_2 : \phi_2)$ as $f_1 > f_2$.

Example 7. *Consider the knowledge base $\mathcal{K}^x_>$, defined via*

$$\mathcal{K}^x_> = \{f_1 : \boldsymbol{G}\neg a, f_2 : \boldsymbol{X}a, f_2 > f_1\}.$$

Then $\mathcal{K}^x_>$ is "consistent" (Intuitively, any interpretation that assigns $a = 1$ in t_1 and $a = 0$ otherwise is a model of $\mathcal{K}^x_>$).

The example shows how knowledge bases with preference relations are non-monotonic in that they preserve *justification*: Even if it is justified that ϕ should hold based on a set of formulas X, there might be a set of formulas Y such that when taking $X \cup Y$, it is no longer justified that ϕ should hold [28]. Use-cases for such non-monotonic reasoning for declarative process specifications are many, for example, modelling exceptions/preferences, reasoning over formulas from different sources (possible with different priority) [39], or reasoning over probabilistic LTL specifications, resp., specifications with uncertainty [31, 40]. In the following, we extend our paraconsistent semantics to be able to consider preferences.

As a first step, we want to establish which formulas should or should not hold, according to the preference relation. For any formula ϕ, this is concluded by means of a tag. Following the four-valued approach in [33, 41] this tag can have one of the following forms:

- $+\Delta\phi$, which is meant to indicate that ϕ is *undisputed* and should therefore hold. This is the case if there are no preferences somehow overruling ϕ.
- $-\Delta\phi$, which is meant to indicate that ϕ is not undisputed.
- $+\delta\phi$, which is meant to indicate that ϕ might be disputed but should still hold as it is defeasibly supported. This is for example the case when a preference r that tries to overrule ϕ is itself overruled by a further preference r'.
- $-\delta\phi$, which is meant to indicate that ϕ is definitely overruled/defeated and should not hold. For example, in Example 7, f_1 is definitely overruled by f_2, as f_2 is undisputed and $f_2 > f_1$.

Especially for $+\delta\phi$, the following observation is in order. Following the proposed semantics for defeasible logic by Nute [28], the formulas of a knowledge base and the corresponding preferences can be viewed as a graph (cf. Definition 7), which we denote as a *preference graph*. Here we apply two assumptions over $\mathcal{K}$:

1. The preference graph is acyclic.
2. For every preference $f_1 > f_2$, there is a set of formulas $\Phi \subseteq \mathcal{K}$ that is consistent under the classical semantics such that $\Phi \cup \{f_1, f_2\}$ is inconsistent.

Point 1 is necessary to avoid unresolvable loops between preferences. For example, if we add $\{f_1 > f_2\}$ to the knowledge base in Example 7, $\mathcal{K}^x_>$ becomes inconsistent again, as we would have a contradictory preference relation $\{f_1 > f_2, f_2 > f_1\}$.

Point 2 is necessary to avoid "unnecessary" preferences, e. g., for a set of formulas $\{f_1 : a, f_2 : \mathbf{X}b, f_1 > f_2\}$, the preference would have no effect.

Before we define the inference rules for the introduced taggings, we introduce the following notation.

A formula ϕ is *undisputed* iff there exist no $f_1, f_2 \in \mathcal{K}$ s.t. $f_2 : \phi$ and $(f_1, f_2) \in >$ (analogously for "disputed" if there exist such a relation). In the latter case where f_2 is disputed, we say that f_1 is a *disputer* of f_2. However, if a disputer f_1 of f_2 is itself disputed by a formula f_0, we say that f_2 is *defended*. Importantly, in line with the semantics proposed in [28], a formula is only tagged $+\delta$ if *all* its disputers are *successfully* defended by other formulas which are either undisputed or themselves successfully defended from all their disputers.

For a knowledge base $\mathcal{K}_> = (K, >)$, the inference rules for these taggings are then defined as follows.

$$
\begin{aligned}
&+\Delta : \phi \text{ is tagged } +\Delta\phi \text{ if}\\
&\qquad \phi \text{ is undisputed}\\[2ex]
&-\Delta : \phi \text{ is tagged } -\Delta\phi \text{ if}\\
&\qquad \phi \text{ is disputed}\\[2ex]
&+\delta : \phi \text{ is tagged } +\delta\phi \text{ if}\\
&\qquad (1) +\Delta\phi, \text{ or}\\
&\qquad (2) \forall \text{disputers } \varphi \text{ of } \phi:\\
&\qquad\quad \exists \rho \in \mathcal{K} \text{ s.t. } \rho \text{ defends } \phi \text{ and } +\delta\rho\\[2ex]
&-\delta : \phi \text{ is tagged } -\delta\phi \text{ if}\\
&\qquad \exists \varphi \text{ s.t. } \varphi \text{ disputes } \phi \text{ and } +\delta\varphi
\end{aligned}
$$

As an Example, recall $\mathcal{K}^x_> = \{f_1 : \mathbf{G}\neg a, f_2 : \mathbf{X}a, f_2 > f_1\}$ from Example 7, with the taggings $+\Delta f_2$ and $-\delta f_1$, as via the preference relation, f_2 is undisputed and f_1 is disputed and not defended.

Example 8. *Consider the knowledge base $\mathcal{K}^a_>$, with*

$$\mathcal{K}^a_> = \{f_1 : a, f_2 : \neg a, f_3 : a \wedge b, f_2 > f_1, f_3 > f_2\}$$

Then $+\Delta a \wedge b, -\delta \neg a, +\delta a$.

The introduced inference rules now allow us to extend the definition of *satisfaction* of a knowledge base as follows. For this, a three-valued LTL$_{\text{ff}}$ interpretation $\hat{\nu}$ satisfies a formula ϕ as before, i. e., iff $\hat{\nu}(\phi, t_0) \in \{1, \mathrm{B}\}$. Also, for a three-valued interpretation $\hat{\nu}$ and set of formulas $\mathcal{K}$, let $\Pi(\mathcal{K})$ denote the minimal number of changes in truth value assignments from 0 or 1 needed s.t. $\hat{\nu} \models^3 K$. Then:

Definition 8. *A three-valued interpretation $\hat{\nu}$ satisfies a set of formulas $\mathcal{K}$ iff*

1. *$\hat{\nu} \models^3 \phi$ for all $\phi \in \mathcal{K}$ with $+\delta\phi$, and*
2. *For the set Φ of all formulas $\{\varphi \in \mathcal{K} | -\delta\varphi\}$, it holds that $\Pi(\Phi)$ cannot be decreased while 1. still holds, i. e., there cannot exist a different assignment (swapping only 1 to 0, or 0 to 1) s.t. $\Pi(\Phi)$ could be decreased and 1. still holds.*

The notion of a *model* is correspondingly extended w.r.t. Definition 8.

Example 9. *We recall $\mathcal{K}^x_> = \{f_1 : \boldsymbol{G}\neg a, f_2 : \boldsymbol{X}a, f_2 > f_1\}$. Then we have that $+\Delta \boldsymbol{X}a$ and $-\delta\, \boldsymbol{G}\neg a$. Consider the following interpretations:*

$\hat{\nu}_1$:	$\hat{\nu}_1(t_0, a) = B$	$\hat{\nu}_1(t_1, a) = B$
$\hat{\nu}_2$:	$\hat{\nu}_2(t_0, a) = B$	$\hat{\nu}_2(t_1, a) = 0$
$\hat{\nu}_3$:	$\hat{\nu}_3(t_0, a) = B$	$\hat{\nu}_3(t_1, a) = 1$
$\hat{\nu}_4$:	$\hat{\nu}_4(t_0, a) = 1$	$\hat{\nu}_4(t_1, a) = B$
$\hat{\nu}_5$:	$\hat{\nu}_5(t_0, a) = 1$	$\hat{\nu}_5(t_1, a) = 0$
$\hat{\nu}_6$:	$\hat{\nu}_6(t_0, a) = 1$	$\hat{\nu}_6(t_1, a) = 1$
$\hat{\nu}_7$:	$\hat{\nu}_7(t_0, a) = 0$	$\hat{\nu}_7(t_1, a) = B$
$\hat{\nu}_8$:	$\hat{\nu}_8(t_0, a) = 0$	$\hat{\nu}_8(t_1, a) = 0$
$\hat{\nu}_9$:	$\hat{\nu}_9(t_0, a) = 0$	$\hat{\nu}_9(t_1, a) = 1$

In light of Definition 8, we have the following:

1. *$\hat{\nu}_1$ is a model as $\boldsymbol{X}a$ is satisfied and there is nothing to change from 0 or 1.*
2. *$\hat{\nu}_2$ is not a model as $\boldsymbol{X}a$ is not satisfied.*
3. *$\hat{\nu}_3$ is a model as $\boldsymbol{X}a$ is satisfied. Also, if $\hat{\nu}_3(t_1, a)$ is swapped to 0, $\boldsymbol{X}a$ is not satisfied anymore.*
4. *$\hat{\nu}_4$ is not a model. While $\boldsymbol{X}a$ is satisfied, we can change the assignment in t_0 to 0 which would reduce $\Pi(\{f_1\})$ by 1. This holds analogously for $\hat{\nu}_5$ and $\hat{\nu}_6$ ($\hat{\nu}_5$ also does not satisfy $\boldsymbol{X}a$).*
5. *$\hat{\nu}_7$ and $\hat{\nu}_9$ are models as $\boldsymbol{X}a$ is satisfied and there can be no assignment swap of 0 or 1 s.t. we reduce $\Pi(\{f_1\})$ (while also maintaining that $\boldsymbol{X}a$ is still satisfied, c.f. $\hat{\nu}_9(t_1, a)$). $\hat{\nu}_8$ is not a model as $\boldsymbol{X}a$ is not satisfied.*

Example 10. *We recall $\mathcal{K}^a_> = \{f_1 : a, f_2 : \neg a, f_3 : a \wedge b, f_2 > f_1, f_3 > f_2\}$, with $+\Delta a \wedge b, -\delta\neg a, +\delta a$. Then*

$$\begin{array}{lll}
\hat{\nu}_1 : & \hat{\nu}_1(t_0, a) = 1 & \hat{\nu}_1(t_0, b) = 1 \\
\hat{\nu}_2 : & \hat{\nu}_2(t_0, a) = B & \hat{\nu}_2(t_0, b) = 1 \\
\hat{\nu}_3 : & \hat{\nu}_3(t_0, a) = 1 & \hat{\nu}_3(t_0, b) = B
\end{array}$$

are models, but

$$\begin{array}{lll}
\hat{\nu}_4 : & \hat{\nu}_4(t_0, a) = 0 & \hat{\nu}_4(t_0, b) = B
\end{array}$$

is not a model.

Some important observations of our defined tagging-based models are given in the following. For this, let $\phi \in \mathcal{K}$ be a formula as before.

Proposition 6 (Coherence)**.** *For any $\mathcal{K}$: $\nexists \phi \in \mathcal{K}$ s.t. $+\delta\phi$ and $-\delta\phi$.*

Proposition 7 (Soundness)**.** *If $+\delta\phi$ then $\mathcal{K} \models \phi$.*

We can now leverage the proposed framework to measure inconsistency in process specifications with preferences.

Example 11. *We recall $\mathcal{K}^x_>$ and the corresponding models discussed above. As our measures are based on finding interpretations that are models and assign B to a minimum number of states, for $m > 0$, we have that $\mathcal{I}^{LTL}_d(\mathcal{K}^x_>) = 0$ as expected, as $\boldsymbol{X}a$ is a desired exception to $\boldsymbol{G}a$. Note that for cases where there are multiple inconsistent—but undisputed—formulas, $\mathcal{I}^{LTL}_d$ would be > 0, because of the unresolved conflicts.*

One observation is that for the semantics with preferences proposed in Definition 8, $\mathcal{I}_d^{LTL}$ and $\mathcal{I}_c^{LTL}$ do not satisfy MO anymore, as this is not applicable for the considered non-monotonic setting (as evidenced in the above examples, *adding* a preference relation can lower the degree of inconsistency).

Given a time-sensitive inconsistency measure, various projections can also be used to gain fine-grained insights about the inconsistency w.r.t. preferences. In the following, we propose two such measures using $\mathcal{I}_d^{LTL}$.

First, we can measure inconsistency in the set of undisputed formulas only. This provides insights into conflicts between formulas that should always hold. This insight could be useful to discover such conflicts as a basis for modelling further preferences.

Definition 9. *For a knowledge base $\mathcal{K}_>$, define the number of open conflicting states as*

$$\#\mathsf{openConflictingStates}(\mathcal{K}_>) = \mathcal{I}_d^{LTL}(\{\phi \in \mathcal{K}_> \mid +\Delta\phi\})$$

Example 12. *We recall $\mathcal{K}_>^x, \mathcal{K}_>^a$, then, for $\mathcal{I}_d^{LTL}$, we have*

$$\#\mathsf{openConflictingStates}(\mathcal{K}_>^x) = 0 \quad \text{and} \quad \#\mathsf{openConflictingStates}(\mathcal{K}_>^a) = 0.$$

For a value > 0, this insight could suggest that there are still inherently conflicting (however, (still) undisputed) formulas, and therefore a modeller could consider revising the specification.

A further interesting detail is how often exceptions actually trigger. This, in a sense, characterizes how "strong" the preference relations are over the sequence of time points. For example, consider again $\mathcal{K}_>^x$ with

$$\mathcal{K}_>^x = \{f_1 : \mathbf{G}\neg a, f_2 : \mathbf{X}a, f_2 > f_1\}$$

and consider

$$\mathcal{K}_>^{x2} = \{f_1 : \mathbf{G}\neg a, f_2 : \mathbf{G}a, f_2 > f_1\}.$$

For $\mathcal{K}_>^x$ the preference relation models an exception that applies to one time point (as otherwise $\mathbf{G}\neg a$ should hold). This can be considered as sensible, e. g., the larger m, the exceptions plays an increasingly minor role. However, in $\mathcal{K}_>^{x2}$, the exception occurs in every state. This could indicate that the specification may need to be revised, as the formula f_1 is effectively useless

($\mathcal{K}^{x2}_{>}$ could be equivalently rewritten simply as $\mathbf{G}a$). A further aspect is that modellers might want to keep the number of states in which exceptions can occur lower, as understanding which rule should hold could be more confusing if exceptions occur in many states.

Definition 10. *For a knowledge base $\mathcal{K}_{>} = (K, >)$, the number of exception states is defined as*

$$\#\mathsf{exceptionStates}(\mathcal{K}_{>}) = \mathcal{I}_d^{LTL}(\mathcal{K}_{>}) - \mathcal{I}_d^{LTL}(\mathcal{K}_{>} \setminus >)$$

Example 13. *We recall $\mathcal{K}^{x}_{>}$ and $\mathcal{K}^{x2}_{>}$. For any $m > 0$, $\#\mathsf{exceptionStates}(\mathcal{K}^{x}_{>}) = 1$ and $\#\mathsf{exceptionStates}(\mathcal{K}^{x2}_{>}) = m$.*

It is worth noting that for the number of exceptions, although not time-sensitive, it could also make sense to follow a formula-based approach, e. g., counting the number of problematic formulas before and after removing the preferences. For example, recalling $\mathcal{K}^{a}_{>} = \{f_1 : a, f_2 : \neg a, f_3 : a \wedge b, f_2 > f_1, f_3 > f_2\}$, we see that $\#\mathsf{exceptionStates}(\mathcal{K}^{a}_{>}) = 1$, but the number of problematic formulas after removing the preference relation is 3, which provides a more granular picture in this case. The choice of a specific approach depends on the use-case; both approaches can be used to assess the importance, or strength, of the contained preferences.

4.3. Element-Based Time-Sensitive Inconsistency Measurement

In the previous subsections, we proposed time-sensitive inconsistency measures for *sets* of LTL$_{\mathrm{ff}}$ formulas. In this subsection we show how to measure the contribution of each formula *individually* to the inconsistency of the set. Let $\mathcal{F}$ be the set of all formulas of LTL$_{\mathrm{ff}}$. Here, we are looking for a *culpability function* C, where $C : \mathbb{K} \times \mathcal{F} \to [0, \infty)$. There are several reasonable ways of measuring the culpability of a formula in a set and we will consider several of these in this section.

Consider $\mathcal{K}_8 = \{\mathbf{G}(a \wedge \neg a); \mathbf{X}b; \mathbf{X}\neg b\}$. Consider also the "classical" culpability measure $\mathcal{C}_{\#}$ [14], which counts the number of minimal inconsistent subsets a formula is contained in. For $\mathcal{K}_8$, we have that $\mathsf{MI}(\mathcal{K}_8) = \{\{\mathbf{G}(a \wedge \neg a)\}, \{\mathbf{X}b, \mathbf{X}\neg b\}\}$. So $\mathcal{C}_{\#}(\mathcal{K}_8, \mathbf{G}(a \wedge \neg a)) = \mathcal{C}_{\#}(\mathcal{K}_8, \mathbf{X}b) = \mathcal{C}_{\#}(\mathcal{K}_8, \mathbf{X}\neg b) = 1$. In the example, all three formulas obtain the same blame value. But for larger m, we see that $\mathbf{G}(a \wedge \neg a)$ affects many more states (and the other formulas affect only one state). This shows that the discussed measure $\mathcal{C}_{\#}$

is again not—as we call it—“time-sensitive”. Ideally, for element-based assessments of LTL$_{\text{ff}}$ formulas, we would also like to be able to distinguish the blame they carry in the context of the overall inconsistency in a time-sensitive way. We therefore propose the following postulates for culpability measures in our setting, the first two adapted from [42], and the third being a variant of *time-sensitivity* defined for culpability measures. For this, let $\mathcal{C}$ be a culpability measure, $\mathcal{K}$ a knowledge base, and $\phi, \varphi \in \mathcal{K}$.

***Consistency*$_{\mathcal{C}}$ ($\mathsf{CO}_{\mathcal{C}}$)** If $\mathcal{K} \not\models \perp$, then $\mathcal{C}(\mathcal{K}, \phi) = 0$ for all $\phi \in \mathcal{K}$.

***Blame*$_{\mathcal{C}}$($\mathsf{BL}_{\mathcal{C}}$)** $\mathcal{C}(\mathcal{K}, \phi) > 0$ for all $\phi \in \mathsf{Problematic}(\mathcal{K})$.

***Time Sensitivity*$_{\mathcal{C}}$($\mathsf{TS}_{\mathcal{C}}$)** If $(\mathcal{I}_d^{LTL}(\mathcal{K}) - \mathcal{I}_d^{LTL}(\mathcal{K} \setminus \{\phi\})) > (\mathcal{I}_d^{LTL}(\mathcal{K}) - \mathcal{I}_d^{LTL}(\mathcal{K} \setminus \{\varphi\}))$, then $\mathcal{C}(\mathcal{K}, \phi) > \mathcal{C}(\mathcal{K}, \varphi)$.

A naive culpability measure in our setting would be to take the marginal contribution each formula has on the value of $\mathcal{I}_d^{LTL}$, i. e., for a knowledge base $\mathcal{K}$ and $\phi \in \mathcal{K}$, define $\mathcal{C}_{MC}$ via

$$\mathcal{C}_{MC}(\mathcal{K}, \phi) = \mathcal{I}_d^{LTL}(\mathcal{K}) - \mathcal{I}_d^{LTL}(\mathcal{K} \setminus \phi)$$

Example 14. *We recall $\mathcal{K}_8 = \{\boldsymbol{G}(a \wedge \neg a); \boldsymbol{X}b; \boldsymbol{X}\neg b\}$ and the introduced measures $\mathcal{C}_{\#}, \mathcal{C}_{MC}$. Then we see that for $m > 0$ we have*

$$\begin{array}{lll} \mathcal{C}_{\#}(\mathcal{K}_8, \boldsymbol{G}(a \wedge \neg a)) = 1 & \mathcal{C}_{\#}(\mathcal{K}_8, \boldsymbol{X}a) = 1 & \mathcal{C}_{\#}(\mathcal{K}_8, \boldsymbol{X}\neg a) = 1 \\ \mathcal{C}_{MC}(\mathcal{K}_8, \boldsymbol{G}(a \wedge \neg a)) = m & \mathcal{C}_{MC}(\mathcal{K}_8, \boldsymbol{X}a) = 0 & \mathcal{C}_{MC}(\mathcal{K}_8, \boldsymbol{X}\neg a) = 0 \end{array}$$

While the example shows that $\mathcal{C}_{MC}$ is time-sensitive (while as established $\mathcal{C}_{\#}$ is not), we see that $\mathcal{C}_{MC}$ does not satisfy BL. This is due to the basis on atoms, and not on formulas. Even if one would define a similar version $\mathcal{C}_{MC'}(\mathcal{K}, \phi) = \mathcal{I}_c^{LTL}(\mathcal{K}) - \mathcal{I}_c^{LTL}(\mathcal{K} \setminus \phi)$ over $\mathcal{I}_c^{LTL}$, this would still not be granular enough, e. g., for $\mathcal{K}_x = \{a \wedge \neg a; a \wedge \neg a \wedge b\}$, we would still get that both formulas would be assigned a value of 0. In turn, both approaches of minimal inconsistent subsets ($\mathcal{C}_{\#}$) and simple marginal contribution ($\mathcal{C}_{MC}$) are not good culpability measures in our setting. In the following, we show how a better culpability measure can be defined using Shapley inconsistency values [10].

Shapley values [43] were introduced originally in game theory where coalitions of players can obtain various payoffs. If there are n players,

$N = \{1, 2, \ldots, n\}$ and the payoff for every coalition C, that is subset of N, is known as $p(C)$, then the Shapley value for each player is its average marginal contribution over all possible coalitions. A commonly used formula for the Shapley value of player i is

$$S(i) = \sum_{C \subseteq N} \frac{(|C|-1)!(n-|C|)!}{n!}(p(C) - p(C \setminus \{i\}))$$

where C is an arbitrary nonempty subset of N.

Shapley values have nice properties and have been used in various fields including economics and machine learning. In our case we are given a set of formulas and want to know the contribution of each formula to the inconsistency measure of the set. Note that in many cases each formula by itself is consistent, and hence has inconsistency measure 0 by itself, and yet in combination with other formulas causes inconsistencies.

We continue by considering the motivational example for time-sensitive, element-based measurement from the beginning of this section. We will use both time sensitive measures we have defined: $\mathcal{I}_d^{LTL}$ and $\mathcal{I}_c^{LTL}$. Instead of using i for the i^{th} player, we write the Shapley value of formula ϕ as $S(\phi)$ (for the inconsistency measure under consideration). To indicate the Shapley value for a formula ϕ using a specific inconsistency measure $\mathcal{I}$, we write $S(\phi, \mathcal{I})$. So given a knowledge base $\mathcal{K}$, we obtain for $\phi \in \mathcal{K}$

$$S(\phi, \mathcal{I}) = \sum_{\mathcal{K}' \subseteq \mathcal{K}} \frac{(|\mathcal{K}'|-1)!(|\mathcal{K}|-|\mathcal{K}'|)!}{|\mathcal{K}|!}(\mathcal{I}(\mathcal{K}') - \mathcal{I}(\mathcal{K}' \setminus \{\phi\}))$$

where $\mathcal{K}'$ is an arbitrary nonempty subset of $\mathcal{K}$.

Example 15. *Recall $\mathcal{K}_8 = \{\boldsymbol{G}(a \wedge \neg a); \boldsymbol{X}b, \boldsymbol{X}\neg b\}$. Here,*
$S(\boldsymbol{G}(a \wedge \neg a), \mathcal{I}_d^{LTL}) = \frac{0! \times 2!}{3!} \times m + \frac{1! \times 1!}{3!} \times m + \frac{1! \times 1!}{3!} \times m + \frac{2! \times 0!}{3!} \times (m-1) = m - \frac{1}{3}$,
$S(\boldsymbol{X}b, \mathcal{I}_d^{LTL}) = S(\boldsymbol{X}\neg b, \mathcal{I}_d^{LTL}) = \frac{1! \times 1!}{3!} \times 1 + \frac{2! \times 0!}{3!} \times 0 = \frac{1}{6}$.
The sum of the Shapley values is m.
For $\mathcal{I}_c^{LTL}$ we obtain the Shapley values as
$S(\boldsymbol{G}(a \wedge \neg a), \mathcal{I}_c^{LTL}) = \frac{0! \times 2!}{3!} \times m + \frac{1! \times 1!}{3!} \times m + \frac{1! \times 1!}{3!} \times m + \frac{2! \times 0!}{3!} \times m = m$,
$S(\boldsymbol{X}b, \mathcal{I}_c^{LTL}) = S(\boldsymbol{X}\neg b, \mathcal{I}_c^{LTL}) = \frac{1! \times 1!}{3!} \times 1 + \frac{2! \times 0!}{3!} \times 1 = \frac{1}{2}$.
Note how the sum of the inconsistency measures of the formulas adds up to the inconsistency measure of the set, a feature of the Shapley value, as it distributes the inconsistency measure of the set to the formulas individually.

So we see that the Shapley approach allows the distribution of the overall "blame" obtained via the time-sensitive inconsistency measures among all formulas of the knowledge base. This allows for reasoning about highly problematic formulas from the perspective of "time-sensitivity", e. g., in the example we see that $\mathbf{G}(a \wedge \neg a)$ is correctly assigned a much higher culpability value than both $\mathbf{X}b$ and $\mathbf{X}\neg b$, which reflects the notion of the number of affected states. Here, the distribution via the Shapley approach also correctly transfers the intuition of the introduced time sensitivity postulate, which we show with the following concluding example.

Example 16. *We recall our main example* $\mathcal{K}_1 = \{\boldsymbol{X}a, \boldsymbol{X}\neg a\}$ *and* $\mathcal{K}_2 = \{\boldsymbol{G}a, \boldsymbol{G}\neg a\}$*. Then we have* $S(\boldsymbol{X}a, \mathcal{I}_d^{LTL}) = S(\boldsymbol{X}\neg a, \mathcal{I}_d^{LTL}) = \frac{1}{2!}(0+0+1) = \frac{1}{2}$*, and* $S(\boldsymbol{G}a, \mathcal{I}_c^{LTL}) = S(\boldsymbol{G}\neg a, \mathcal{I}_c^{LTL}) = \frac{1}{2!}(0+0+m) = \frac{m}{2}$ *(the same for* $\mathcal{I}_c^{LTL}$*).*

The following example illustrates Shapley values in a case where the inconsistency measure is infinity.

Example 17. *Let* $\mathcal{K} = \{\boldsymbol{X}(a \wedge \neg a), \boldsymbol{X}\boldsymbol{X}b, \boldsymbol{X}\ldots\boldsymbol{X}\neg b\}$ *where by* $\boldsymbol{X}\ldots\boldsymbol{X}\neg b$*, which rewrite as* ϕ*, we mean* $m+1$ $\boldsymbol{X}$*s in front of* $\neg b$*. According to our definition,* $\mathcal{I}_c^{LTL}(\{\phi\}) = \mathcal{I}_d^{LTL}(\{\phi\}) = \infty$*. Then we have*
$S(\boldsymbol{X}(a \wedge \neg a), \mathcal{I}_d^{LTL}) = S(\boldsymbol{X}(a \wedge \neg a), \mathcal{I}_c^{LTL}) = \frac{0! \times 2!}{3!} \times 1 + \frac{1! \times 1!}{3!} \times 1 + 0 + 0 = \frac{1}{2}$*,*
$S(\boldsymbol{X}\boldsymbol{X}b, \mathcal{I}_d^{LTL}) = S(\boldsymbol{X}\boldsymbol{X}b, \mathcal{I}_c^{LTL}) = 0$ *and using the fact that any positive number times infinity is infinity and the sum of infinities is infinity, we obtain* $S(\phi, \mathcal{I}_d^{LTL}) = S(\phi, \mathcal{I}_c^{LTL}) = \infty$*.*
The sum of all the Shapley values is $\infty + \frac{1}{2} = \infty$ *which is the inconsistency value of* $\mathcal{K}$*. Note also that even in this case where one formula has inconsistency value infinity, the Shapley value distinguishes the inconsistent formula* $a \wedge \neg a$ *from the free formula* $\boldsymbol{X}\boldsymbol{X}b$*.*

Consider now the case where we add a second formula whose inconsistency is also infinity. Let $\mathcal{K}' = \{\boldsymbol{X}(a \wedge \neg a), \boldsymbol{X}\boldsymbol{X}b, \boldsymbol{X}\ldots\boldsymbol{X}\neg b, \boldsymbol{X}\ldots\boldsymbol{X}\boldsymbol{X}\neg b\}$*. So the new formula is* $\boldsymbol{X}\phi$*. Again,* $\mathcal{I}_c^{LTL}(\{\boldsymbol{X}\phi\}) = \mathcal{I}_d^{LTL}(\{\boldsymbol{X}\phi\}) = \infty$*. Then we have*
$S(\boldsymbol{X}(a \wedge \neg a), \mathcal{I}_d^{LTL}) = S(\boldsymbol{X}(a \wedge \neg a), \mathcal{I}_c^{LTL}) = \frac{0! \times 3!}{4!} \times 1 + \frac{1! \times 1!}{4!} \times 1 + 0 + 0 = \frac{7}{24}$*.*
$S(\boldsymbol{X}\boldsymbol{X}b, \mathcal{I}_d^{LTL}) = S(\boldsymbol{X}\boldsymbol{X}b, \mathcal{I}_c^{LTL}) = 0$*, as before. Note how with the extra formula, the Shapley value of* $a \wedge \neg a$ *became smaller. The other thing that is different now is that in the computation of* $S(\phi, \mathcal{I}_d^{LTL})$*,* $S(\phi, \mathcal{I}_c^{LTL})$*,* $S(\boldsymbol{X}\phi, \mathcal{I}_d^{LTL})$*, and* $S(\boldsymbol{X}\phi, \mathcal{I}_c^{LTL})$*, we have cases of* $\infty - \infty$*, such as when we compute* $\mathcal{I}_d^{LTL}(\{\phi, \boldsymbol{X}\phi\}) - \mathcal{I}_d^{LTL}(\{\boldsymbol{X}\phi\})$*. But the value we give to the subtraction of infinities makes no difference; it can be any number, such as* 0*. The reason is that* $\mathcal{I}_d^{LTL}(\{\phi\}) - \mathcal{I}_d^{LTL}(\emptyset) = \infty$*, and that is part of the*

computation of $S(\phi, \mathcal{I}_d^{LTL})$. The same holds for $S(\phi, \mathcal{I}_c^{LTL})$, $S(\boldsymbol{X}\phi, \mathcal{I}_d^{LTL})$, and $S(\boldsymbol{X}\phi, \mathcal{I}_c^{LTL})$. So we obtain $S(\phi, \mathcal{I}_d^{LTL}) = S(\phi, \mathcal{I}_c^{LTL}) = S(\boldsymbol{X}\phi, \mathcal{I}_d^{LTL}) = S(\boldsymbol{X}\phi, \mathcal{I}_c^{LTL}) = \infty$.

Proposition 8. *Let $\mathcal{I}$ be an inconsistency measure that satisfies CO and TS. Then $S(\phi, \mathcal{I})$ satisfies $\mathsf{CO}_{\mathcal{C}}$, $\mathsf{BL}_{\mathcal{C}}$, and $\mathsf{TS}_{\mathcal{C}}$.*

While we have focused on measures for individual *formulas*, the introduction of a *preference relation* also gives rise to some element-based questions w.r.t. it. For example, if the modeller could be provided with measures to assess the importance of individual preferences, this could be useful in the scope of re-modelling, e. g., for assessing the possible impact of deleting a preference. We propose some ideas for such element-based measures in the following.

First, for any formula ϕ in the knowledge base that might be overruled by other formulas, the modeller might want to know by how many exceptions ϕ is actually overruled. This could be useful for assessing how disputed an individual formula is, or, whether a formula will always hold. Note that this is not necessarily equal to the number of preferences related to ϕ, as one has to take possible defenders into account to (i. e., a formula which is successfully defended against all its disputees would have a value of 0).

Definition 11. *For a knowledge base $\mathcal{K}_> = (K, >)$ and $\phi \in \mathcal{K}$, the number of disputees is defined via*

$$\#\mathsf{disputees}(\mathcal{K}_>, \phi) = min\{|X| \mid X \subseteq > \text{ and } +\delta\phi \text{ for } (K, > \backslash X)\}$$

The intuition of $\#\mathsf{disputees}$ is that a higher value reflects a higher dispute of a formula ϕ, i. e., more preference relations/exceptions would need to be deleted for ϕ to be assumed to hold. A formula that is undisputed would have a minimal value of 0. From the opposite viewpoint, it could be useful to analyze for a formula ϕ (which may currently be assumed to hold) how many defending preference relations must be deleted such that ϕ will be tagged $-\delta$. The intuition would be that a higher value would indicate a higher robustness of the formula, e. g., a formula would be more robust if it has more defenders. In case a formula is undisputed, it is assigned ∞, indicating maximal robustness, as it must hold in any case and even without preference information.

Definition 12. *For a knowledge base $\mathcal{K}_> = (K, >)$ and $\phi \in \mathcal{K}$, the robustness of ϕ is defined via*

$$\textsf{robustness}(\mathcal{K}_>, \phi) = min\{|X| \mid X \subseteq > \text{ and } -\delta\phi \text{ for } (K, > \setminus X)\},$$
$$\text{or } \infty \text{ if } +\Delta\phi$$

From the perspective of modeling preferences, it would also be useful to assess how many inconsistencies will be introduced, should a preference be deleted. This would be useful for assessing the importance of individual preferences.

Definition 13. *For a knowledge base $\mathcal{K}_> = (K, >)$ and $r \in >$, the number of new affected states (due to deletion of r) is defined via*

$$\#\textsf{newAffectedStates}(\mathcal{K}_>, r) = \mathcal{I}_d^{LTL}(\mathcal{K}_> \setminus \{r\}) - \mathcal{I}_d^{LTL}(\mathcal{K}_>)$$

Example 18. *Consider the knowledge base $\mathcal{K}^b_>$, defined via*

$$\mathcal{K}^b_> = \{f_1 : \boldsymbol{X}a, f_2 : \boldsymbol{G}\neg a, f_3 : \boldsymbol{G}b, f_4 : \boldsymbol{G}\neg b, f_1 > f_2, f_3 > f_4\}.$$

First, we see that f_1, f_3 are maximally robust *(∞), and f_2, f_4 each have 1* disputee*. Furthermore, by deleting $r = f_1 > f_2$, $\#\textsf{newAffectedStates}(\mathcal{K}_>, r)$ would be 1, however, for $r' = f_3 > f_4$, $\#\textsf{newAffectedStates}(\mathcal{K}_>, r') = m$. So we see that deleting $f_3 > f_4$ would have a larger, what we call time-sensitive, impact than for $f_1 > f_2$.*

An analogous measure could be defined for how many inconsistent states are resolved after *adding* a specific preference, e.g., to assess the effectiveness of a possible addition to the set of preferences.

5. Application to Declarative Process Models

A common application scenario for LTL$_f$ is that of declarative process models [44], which are sets of (LTL$_f$-based) constraints. For such declarative process models, the issue of inconsistency is equally as problematic, as any inconsistencies between the constraints make the declarative process model unsatisfiable. We therefore show how our results can be applied in use-cases relying on such declarative process models, in particular, the Declare standard [44].

Note that while there have been some of works addressing the issue of inconsistency in declarative process models in general [4, 23, 5], those works however do not allow looking "into" (minimally) inconsistent sets or assess their severity. The means presented in this work therefore extend the existing results and allow for a direct application to declarative process models, which we present in the following.

An important distinction is that some variants of Declare implement interpretations as "simple traces". In our terms, these are essentially interpretations where at every point in time only one atom may be set to true. This can make sense in many real-life domains, such as business process management, where activities may be assumed to occur in an ordered manner. As an abbreviation, sequences such as *"abc"* are often used to denote a simple trace, in this case meaning that a holds in t_0, b holds in t_1, and c holds in t_2. Formally, the only difference of using simple traces as opposed to classical interpretations (as we have used them) is that the set of models is restricted by an additional constraint. This does not change the definition of the introduced measures in any way. An aspect to keep in mind however is that knowledge bases such as $\{\mathbf{X}a, \mathbf{X}b\}$ would always be inconsistent over simple traces (which is not the case for the classical interpretations as used in this work where multiple statements may hold at the same point in time). Note also that this does not affect Proposition 3, as (given the length assumptions via Proposition 3) setting the assignment of all atoms to B still satisfies all formulas, even under the "simple trace" view.

5.1. Inconsistency Measurement in Declarative Process Models

A declarative process model consists of a set of constraints. Typically, these constraints are constructed using predefined templates, i. e., predicates, that are specified relative to a set of propositions (e. g., company activities).

Definition 14 (Declarative Process Model)**.** *A declarative process model is a tuple* $\boldsymbol{M} = (\boldsymbol{A}, \boldsymbol{T}, \boldsymbol{C})$*, where* $\boldsymbol{A}$ *is a set of propositions,* $\boldsymbol{T}$ *is a set of constraint types, and* $\boldsymbol{C}$ *is the set of constraints, which instantiate the template elements in* $\boldsymbol{T}$ *with activities in* $\boldsymbol{A}$*.*[6]

In this work, we consider the declarative modeling language Declare [44], which offers a set of "standard" templates. We will use the selection of

[6]For readability, we will denote declarative process models as a set of constraints (**C**)

templates shown in Table 2. We refer the reader to [4] for an overview of other Declare template types and the corresponding semantics.

Template	**$\mathrm{LTL}_{\mathrm{ff}}$ Semantics**
$\textsc{Init}(a)$	a
$\textsc{End}(a)$	$\mathbf{G}(a \vee \mathbf{F}a)$
$\textsc{Response}(a,b)$	$\mathbf{G}(a \rightarrow \mathbf{F}b)$
$\textsc{NotResponse}(a,b)$	$\mathbf{G}(a \rightarrow \neg\mathbf{F}b)$
$\textsc{ChainResponse}(a,b)$	$\mathbf{G}(a \rightarrow \mathbf{X}b)$
$\textsc{NotChainResponse}(a,b)$	$\mathbf{G}(a \rightarrow \neg\mathbf{X}b)$
$\textsc{AtLeast}(a,n)$	$\mathbf{F}(a \wedge \mathbf{X}(\textsc{atLeast}(a, n\text{-}1)))$,
	$\textsc{atLeast}(a,1) = \mathbf{F}(a)$
$\textsc{AtMost}(a,n)$	$\mathbf{G}(\neg a \vee \mathbf{X}(\textsc{atMost}(a, n-1)))$,
	$\textsc{atMost}(a,0) = \mathbf{G}(\neg a)$

Table 2: $\mathrm{LTL}_{\mathrm{ff}}$ Semantics for a selection of Declare templates.

By viewing the constraints of a declarative process model as equivalent $\mathrm{LTL}_{\mathrm{ff}}$ formulas (see above), our approach for measuring inconsistency in $\mathrm{LTL}_{\mathrm{ff}}$ can be applied to Declare in a straightforward manner. In the following examples, we will show the results both for $\mathcal{I}_c^{LTL}$ and $\mathcal{I}_d^{LTL}$ and observe how, in some cases, $\mathcal{I}_c^{LTL}$ makes a finer distinction than $\mathcal{I}_d^{LTL}$.

Example 19. *Consider the sets of constraints C_a and C_b, defined via*

$$\begin{aligned}
C_a &= \{\textsc{Init}(a), \textsc{Response}(a,b), \textsc{NotResponse}(a,b)\}\\
&\quad (\Leftrightarrow \{a, \boldsymbol{G}(a \rightarrow \boldsymbol{F}b), \boldsymbol{G}(a \rightarrow \neg\boldsymbol{F}b)\})\\
C_b &= \{\textsc{Init}(a), \textsc{Response}(a,b), \textsc{NotResponse}(a,b),\\
&\qquad \textsc{Response}(a,c), \textsc{NotResponse}(a,c)\}\\
&\quad (\Leftrightarrow \{a, \boldsymbol{G}(a \rightarrow \boldsymbol{F}b), \boldsymbol{G}(a \rightarrow \neg\boldsymbol{F}b), \boldsymbol{G}(a \rightarrow \boldsymbol{F}c), \boldsymbol{G}(a \rightarrow \neg\boldsymbol{F}c)\}).
\end{aligned}$$

We obtain $\mathcal{I}_c^{LTL}(C_a) = \mathcal{I}_d^{LTL}(C_a) = 1$, while $\mathcal{I}_c^{LTL}(C_b) = 2$ but $\mathcal{I}_d^{LTL}(C_b) = 1$ (for any $m > 0$).

Due to the recursive definition of some "existence" constraints (cf. Table 2), note that also inconsistencies concerned with cardinalities can be assessed correctly.

Example 20. *Consider the sets of constraints C_c and C_d, defined via*[7]
$C_c = \{\textsc{AtMost}(a, 1),\ \textsc{AtLeast}(a, 2)\}$ *and*
$C_d = \{\textsc{AtMost}(a, 1),\ \textsc{AtLeast}(a, 100)\}$*, assuming that $m \geq 99$. Then* $\mathcal{I}_c^{LTL}(C_c) = \mathcal{I}_d^{LTL}(C_c) = 1 < \mathcal{I}_c^{LTL}(C_d) = \mathcal{I}_d^{LTL}(C_d) = 99$.

As a border case, note that any inconsistency referring to a point in time beyond the assumed sequence of states will return a value of ∞ per definition, as we cannot assess any error that leaves the boundaries of our logical framework.

Example 21. *Let $C_e = \{\textsc{End}(a), \textsc{ChainResponse}(a, b)\}$. In this case* $\mathcal{I}_c^{LTL}(C_e) = \mathcal{I}_d^{LTL}(C_e) = \infty$.

In addition to inconsistency measurement, a central contribution of Section 4 was the introduction of defeasible reasoning capabilities. To apply this to Declare, we extend the Definition of a declarative process model as follows.

Definition 15 (Declarative Process Model with Preferences)**.** *A declarative process model with preferences is a tuple $\boldsymbol{M} = (\boldsymbol{A}, \boldsymbol{T}, \boldsymbol{C}, >)$, where $\boldsymbol{A}$ is a set of propositions, $\boldsymbol{T}$ is a set of constraint types, and $\boldsymbol{C}$ is the set of constraints, which instantiate the template elements in $\boldsymbol{T}$ with activities in $\boldsymbol{A}$, and $> \subseteq \boldsymbol{C} \times \boldsymbol{C}$ is a preference relation.*

This allows for the incorporation of the results presented on inconsistency measurement with preferences to Declare. As stated, there can be many use-cases for this, in particular, any setting where knowledge can change over time, or where potential conflicts among declarative constraints must be handled in a flexible manner. For instance, it is a widely acknowledged problem that current declarative discovery techniques may yield sets of constraints that can be conflicting w.r.t. to certain traces (or even inconsistent), as state-of-the-art discovery techniques are mostly based on the support of individual constraints and fail to consider all interrelations of constraints [4]. Here, preferences can be used to model exceptions without the need to remodel any constraints themselves. Also, in case data perspectives are also considered, e. g., MP-Declare [45], preferences allow for modeling default rules and exceptions depending on context data.

[7]The equivalent LTL_{ff} formulas can be constructed as shown in Table 2 by recursively nesting the"$\mathsf{AtLeast}$" constraint (n-times) until its end condition $\mathsf{AtLeast}(a, 1) = \mathbf{F}a$. The concrete formula is omitted here due to excessive length.

Example 22. *Consider the set of constraints C_f with*

$$C_f = \{\textsc{Response}(a,c), \textsc{NotResponse}(b,c)\},$$

and assume the preference $\textsc{Response}(a,c) > \textsc{NotResponse}(b,c)$*. The provided preference can be applied to specify an exception of the second constraint, here, whenever a occurs before b (without being resolved by c beforehand). To clarify, consider the following interpretation $\hat{\nu}_1$:*

$$\begin{array}{lll} \hat{\nu}_1(t_0,a) = 1 & \hat{\nu}_1(t_0,b) = 1 & \hat{\nu}_1(t_0,c) = 0 \\ \hat{\nu}_1(t_1,a) = 0 & \hat{\nu}_1(t_1,b) = 0 & \hat{\nu}_1(t_1,c) = 1 \end{array}$$

Here, the preference information allows us to correctly reason that $\hat{\nu}_1$ satisfies C_f, as the $\textsc{NotResponse}(b,c)$ *is overruled. In terms of "simple traces", this would be comparable to allowing the trace "abc" (which, without the preference information, would not be accepted by C_f).*

5.2. Element-Based Inconsistency Measurement in Declarative Process Models

Our proposed results on element-based measurement can also be applied for Declare, e. g., for pin-pointing highly problematic elements. In the following, we highlight some concrete use-cases for element-based analysis in Declare.

As an initial example, consider again $C_a = \{\textsc{Init}(a), \textsc{Response}(a,b), \textsc{NotResponse}(a,b)\}$. In this case all three constraints are needed for the inconsistency. This means that the Shapley value must allocate the inconsistency measure 1 equally to the three constraints. Thus, $S(\textsc{Init}(a), \mathcal{I}_c^{LTL}) = S(\textsc{Response}(a,b), \mathcal{I}_c^{LTL}) = S(\textsc{NotResponse}(a,b), \mathcal{I}_c^{LTL}) = S(\textsc{Init}(a), \mathcal{I}_d^{LTL}) = S(\textsc{Response}(a,b), \mathcal{I}_d^{LTL}) = S(\textsc{NotResponse}(a,b), \mathcal{I}_d^{LTL}) = \frac{1}{3}$. As we next show, in more complex examples, our approach yields a more fine-grained insight.

Example 23. *We recall the constraint set C_b:*

$$\begin{aligned} C_b = \{&\textsc{Init}(a), \textsc{Response}(a,b), \textsc{NotResponse}(a,b), \\ &\textsc{Response}(a,c), \textsc{NotResponse}(a,c)\} \\ &(\Leftrightarrow \{a, \boldsymbol{G}(a \to \boldsymbol{F}b), \boldsymbol{G}(a \to \neg\boldsymbol{F}b), \boldsymbol{G}(a \to \boldsymbol{F}c), \boldsymbol{G}(a \to \neg\boldsymbol{F}c)\}) \end{aligned}$$

We start with the computation for $\mathcal{I}_c^{LTL}$. Recall that $\mathcal{I}_c^{LTL}(C_b) = 2$. We obtain $S(\textsc{Init}(a), \mathcal{I}_c^{LTL}) = \frac{2\times4+4\times6+2\times24}{120} = \frac{2}{3}$. It is clear that the other *four constraints play equivalent roles and must have the same Shapley values. In fact,* $S(\textsc{Response}(a,b),\mathcal{I}_c^{LTL}) = S(\textsc{NotResponse}(a,b),\mathcal{I}_c^{LTL}) = S(\textsc{Response}(a,c),\mathcal{I}_c^{LTL}) = S(\textsc{NotResponse}(a,c),\mathcal{I}_c^{LTL}) = \frac{1\times4+2\times6+24}{120} = \frac{1}{3}$.

Next we consider $\mathcal{I}_d^{LTL}$ *for this example, recalling that* $\mathcal{I}_d^{LTL}(C_b) = 1$. *We obtain* $S(\textsc{Init}(a),\mathcal{I}_d^{LTL}) = \frac{2\times4+4\times6+24}{120} = \frac{7}{15}$. *Again, the other four constraints have the same Shapley values:* $S(\textsc{Response}(a,b),\mathcal{I}_d^{LTL}) = S(\textsc{NotResponse}(a,b),\mathcal{I}_d^{LTL}) = S(\textsc{Response}(a,c),\mathcal{I}_d^{LTL}) = S(\textsc{NotResponse}(a,c),\mathcal{I}_d^{LTL}) = \frac{4+2\times6}{120} = \frac{2}{15}$. *It is interesting to note that for* $\mathcal{I}_c^{LTL}$, $\textsc{Init}(a)$ *has twice the Shapley value of each of the other four constraints, while for* $\mathcal{I}_d^{LTL}$, $\textsc{Init}(a)$ *gets almost half of the total inconsistency measure.*

So we can correctly infer that $\textsc{Init}(a)$ carries the highest blame. For this example, the same holds for an MI-based approach (e. g., $C_\#$). The next example shows that our time-sensitive inconsistency measures give more appropriate values in some cases.

Example 24. *Consider the constraint set* C_f*, defined via*

$$C_f = \{\textsc{AtMost}(a,1), \textsc{AtLeast}(a,10), \textsc{AtMost}(b,1), \textsc{AtLeast}(b,2)\}$$

Here we have that

$$\mathsf{MI}(C_f) = \{\{\textsc{AtMost}(a,1), \textsc{AtLeast}(a,10)\}, \{\textsc{AtMost}(b,1), \textsc{AtLeast}(b,2)\}\}$$

With an MI*-based approach such as* $C_\#$*, we would have that*

$$C_\#(\textsc{AtMost}(a,1)) = 1 \qquad C_\#(\textsc{AtLeast}(a,10)) = 1$$
$$C_\#(\textsc{AtMost}(b,1)) = 1 \qquad C_\#(\textsc{AtLeast}(b,2)) = 1.$$

Thus the blame of these formulas is not distinguishable in this case. Now we show that our approach can distinguish between the blame of the constraints. Assume that $m \geq 9$. *Then we obtain* $\mathcal{I}_c^{LTL}(C_f) = 10$*, while* $\mathcal{I}_d^{LTL}(C_f) = 9$. *Next we compute the Shapley values. In this case,* $\textsc{AtMost}(a,1)$ *and* $\textsc{AtLeast}(a,10)$ *have the same Shapley values. In the same way,* $\textsc{AtMost}(b,1)$ *and* $\textsc{AtLeast}(b,2)$ *also have the same Shapley values. We obtain*

$$S(\textsc{AtMost}(a,1),\mathcal{I}_c^{LTL}) = S(\textsc{AtLeast}(a,10),\mathcal{I}_c^{LTL}) = \frac{9}{12} + \frac{18}{12} + \frac{9}{4} = 4.5.$$
$$S(\textsc{AtMost}(b,1),\mathcal{I}_c^{LTL}) = S(\textsc{AtLeast}(b,2),\mathcal{I}_c^{LTL}) = \frac{1}{12} + \frac{2}{12} + \frac{1}{4} = 0.5.$$

$$S(\textsc{AtMost}(a,1),\mathcal{I}_d^{LTL}) = S(\textsc{AtLeast}(a,10),\mathcal{I}_d^{LTL}) = \frac{9}{12}+\frac{18}{12}+\frac{8}{4} = 4.25.$$
$$S(\textsc{AtMost}(b,1),\mathcal{I}_d^{LTL}) = S(\textsc{AtLeast}(b,2),\mathcal{I}_d^{LTL}) = \frac{1}{12}+\frac{2}{12} = 0.25.$$

In both cases, the first two formulas are much more problematic than the last two formulas.

These examples show that our approach can provide detailed insights on the severity of inconsistency in declarative process models. Such insights can prove useful for prioritizing or re-modeling different issues of the process specification. In this context, it seems intuitive that conflicts affecting only the next state ($\mathbf{X}$) should be considered as less severe than conflicts affecting multiple following states ($\mathbf{G}$), i. e., for any LTL$_{\mathrm{ff}}$ formula φ, $\mathcal{I}(\{\mathbf{G}\varphi, \mathbf{G}\neg\varphi\}) > \mathcal{I}(\{\mathbf{X}\varphi, \mathbf{X}\neg\varphi\})$.

6. Algorithmic Approaches

In the following, we present approaches based on Answer Set Programming (ASP) for solving the problem of retrieving the value for the presented inconsistency measures wrt. a given knowledge base $\mathcal{K}$. The implementation can be found online[8]. As shown in Section 4.3, the presented element-based measures can be obtained by a straightforward extension building on the corresponding inconsistency measures. Therefore, in the following, we focus on the core problem of retrieving the values for the underlying inconsistency measures, i. e., $\mathcal{I}_d^{LTL}$, and $\mathcal{I}_c^{LTL}$.

ASP has already been applied to compute inconsistency values in propositional logic. In particular, the contension inconsistency measure has been addressed in previous work by the authors [46, 35, 47, 48]. We therefore use the ASP encoding for the contension measure proposed in [47] as a basis for encoding $\mathcal{I}_c^{LTL}$ and $\mathcal{I}_d^{LTL}$.

To begin with, we need to represent the given knowledge base $\mathcal{K}$ itself. We first add a fact for each formula $\varphi \in \mathcal{K}$:

$$\texttt{kbElement(}\varphi\texttt{).} \tag{B1}$$

Note that the φ in $\mathcal{K}$ and the φ in `kbElement/1` are not formally the same. In $\mathcal{K}$, it is an LTL formula, and in `kbElement/1` it is an ASP constant

[8]`https://github.com/aig-hagen/inconsistency-measurement-LTL`

representing that formula. Each $\varphi \in \mathcal{K}$ is "translated" to such a (uniquely defined) constant, i. e., a string starting with a lowercase letter. For instance, a formula φ_1 could be represented as `phi_1` in the corresponding ASP encoding. This concept is extended to subformulas, atoms, and truth values. For example, every conjunction $\varphi = \varphi_1 \wedge \varphi_2$ is represented as

$$\texttt{conjunction(}\varphi\texttt{,}\varphi_1\texttt{,}\varphi_2\texttt{).} \tag{B2}$$

In the same fashion, we define for each disjunction $\varphi = \varphi_1 \vee \varphi_2$ a fact $\texttt{disjunction(}\varphi\texttt{,}\varphi_1\texttt{,}\varphi_2\texttt{)}$ (B3), and for each negated formula $\varphi = \neg\varphi_1$ a fact $\texttt{negation(}\varphi\texttt{,}\varphi_1\texttt{)}$ (B4). In addition, we need to represent those formulas which contain LTL$_{\mathrm{ff}}$-specific operators. Formulas including the *next* operator, i. e., formulas of the form $\varphi = \mathbf{X}\varphi_1$, are encoded as

$$\texttt{next(}\varphi\texttt{,}\varphi_1\texttt{).} \tag{B5}$$

Occurrences of the *until* operator, i. e., formulas of the form $\varphi = \varphi_1\mathbf{U}\varphi_2$, are encoded as

$$\texttt{until(}\varphi\texttt{,}\varphi_1\texttt{,}\varphi_2\texttt{).} \tag{B6}$$

Although not strictly necessary, we also encode the *globally* operator and the *eventually* operator.

Hence, a formula $\varphi = \mathbf{G}\varphi_1$ is represented as

$$\texttt{globally(}\varphi\texttt{,}\varphi_1\texttt{).} \tag{B7}$$

and a formula $\varphi = \mathbf{F}\varphi_1$ is represented as

$$\texttt{eventually(}\varphi\texttt{,}\varphi_1\texttt{).} \tag{B8}$$

Further, we need to encode (sub)formulas which consist of individual atoms. Hence, for each formula φ, which is equal to an atom x, we define

$$\texttt{formulaIsAtom(}\varphi\texttt{,}x\texttt{).} \tag{B9}$$

Moreover, each atom $a \in \mathsf{At}(\mathcal{K})$ is modeled as

$$\texttt{atom(}a\texttt{).} \tag{B10}$$

Additionally, we need to model the truth values of three-valued logic:

```
tv(t;f;b).                                    (B11)
```

By now, we modeled the formulas of the given knowledge base in ASP; however, we still need to represent the sequence of states $T = (t_0, \ldots, t_m)$. To achieve this, we first represent the final state m as

`finalState(`m`).` (B12)

and then define each individual state as follows:

```
state(0..M) :- finalState(M).                 (B13)
```

The "`..`" is an interval operator indicating that each number between 0 and `M` represents a state. Thus, state t_0 corresponds to `0`, state t_1 to `1`, and so forth.

In order to encode the evaluation of formulas, we first need to model that each atom is evaluated to one truth value per state. A cardinality constraint with both the lower and the upper bound set to 1 is used to represent that *exactly* one truth value must be selected per atom and state:

```
1{truthValue(A,S,T) : tv(T)}1 :- atom(A), state(S).     (B14)
```

The above rule is essentially the same as rule (A8) from [47], however, since we are dealing with LTL$_{\text{ff}}$, we need to additionally refer to a state `S`. The same applies to the rule modeling formulas consisting of individual atoms as well as the rules modeling the classical connectives ($\wedge$, $\vee$, $\neg$)—they each correspond to a rule from [47], but additionally contain a connection to a state. Thus, in case a (sub-)formula consists of only a single atom, it must have the same truth value in a given state:

```
truthValue(F,S,T) :- formulaIsAtom(F,G),
                     state(S),
                     truthValue(G,S,T),
                     tv(T).                   (B15)
```

In order for a conjunction to be true in a state, both of its conjuncts have to be true:

```
truthValue(F,S,t) :- conjunction(F,G,H),
                     state(S),
                     truthValue(G,S,t),
                     truthValue(H,S,t).          (B16)
```

For a conjunction to be false, it is sufficient if only one of its conjuncts is false:

```
truthValue(F,S,f) :- conjunction(F,G,H),
                     state(S),
                     1{truthValue(G,S,f);
                          truthValue(H,S,f)}.    (B17)
```

Finally, a conjunction is both if it is neither true nor false:

```
truthValue(F,S,b) :- conjunction(F,_,_),
                     state(S),
                     not truthValue(F,S,t),
                     not truthValue(F,S,f).      (B18)
```

In the same fashion, we can define that a disjunction is false if both of its disjuncts are false (B19), and true if at least one of its disjuncts is true (B20). Again, if a disjunction is neither true nor false, it must be both (B21). Furthermore, a negation is true in a state in three-valued logic if its base formula is false (B22). Analogously, a negation is false if its base formula is true (B23), and a negation is both if its base formula is both as well (B24).

Next, we describe the LTL$_{\text{ff}}$-specific operators. As per Section 3.2, a formula $\varphi = \mathbf{X}\varphi_1$ evaluates to the same truth value in t_i as φ_1 in t_{i+1} if

$i < m$:

```
truthValue(F,S_i,T) :- next(F,G),
                       state(S_i),
                       tv(T),
                       S_j = S_i + 1,
                       S_i < M,
                       finalState(M),
                       truthValue(G,S_j,T).       (B25)
```

Note that the fact that each state is defined as an integer number in ASP (see (B13)) allows us to use arithmetic and comparison operators. Moreover, $\varphi = \mathbf{X}\varphi_1$ evaluates to false in case $i \geq m$:

```
truthValue(F,S_i,f) :- next(F,_),
                       state(S_i),
                       S_i >= M,
                       finalState(M).             (B26)
```

Following again Section 3.2, a formula $\varphi = \varphi_1 \mathbf{U} \varphi_2$ is defined to be true in state t_i if both φ_1 and φ_2 are true in t_i or if there is a $j \in \{i+1, \ldots, m\}$ such that φ_1 is true in all $t_i, \ldots, t_{j-1}$, and φ_2 is true in t_j. To represent the first case in ASP, we add the following rule:

```
truthValue(F,S_i,t) :- until(F,G,H),
                       state(S_i),
                       truthValue(G,S_i,t),
                       truthValue(H,S_i,t).       (B27)
```

We represent the second case by using a cardinality constraint that requires φ_1 to be true in `X` states, with `X` being defined as $j - i$. To ensure that exactly the states $t_i, \ldots, t_{j-1}$ are selected, we formulate two conditions within the cardinality constraint: first, each selected state must be greater than or equal to i, and second, each such state must be strictly smaller than j. In addition, the rule separately states that φ_2 must be true in t_j.

```
truthValue(F,S_i,t) :- until(F,G,H),
                       state(S_i),
                       state(S_j),
                       S_j > S_i,
                       S_j <= M,
                       finalState(M),
                       X{truthValue(G,S,t):  state(S),
                              S >= S_i,
                              S < S_j}X,
                       X = S_j - S_i,
                       truthValue(H,S_j,t).                (B28)
```

The formula $\varphi = \varphi_1 \mathbf{U} \varphi_2$ is evaluated to both in state t_i if there is a $j \in \{i+1, \ldots, m\}$ such that φ_1 is true *or both* in all $t_i, \ldots, t_{j-1}$, and φ_2 is true *or both* in t_j, i. e., $\{\hat{\nu}(t_i, \varphi_1), \ldots, \hat{\nu}(t_{j-1}, \varphi_1), \hat{\nu}(t_j, \varphi_2)\} = \{1, \mathrm{B}\}$. As in the previous rule, we make use of a cardinality constraint. However, since both φ_1 and φ_2 are allowed to be true *or* both in each corresponding state, we need to formulate it a bit differently. To begin with, we do not handle t_j separately, but include it in the cardinality constraint, which leads to its upper and lower bound to be defined by $j - i + 1$ (instead of $j - i$, as before). The cardinality constraint intuitively contains a choice of the following elements, of which exactly $j - i + 1$ must be true:

- φ_1 can be 1 in each $t_i, \ldots, t_{j-1}$
- φ_1 can be B in each $t_i, \ldots, t_{j-1}$
- φ_2 can be 1 in t_j
- φ_2 can be B in t_j

Since each atom in each state has a unique truth value (due to (B14)), we prevent the same formula in the same state from being, e. g., 1 and B at the same time. Hence, exactly the states $t_i, \ldots, t_j$ are selected, and φ_1 and φ_2 are 1 or B in each respective state. Furthermore, we enforce implicitly that at least one element in $\{\hat{\nu}(t_i, \varphi_1), \ldots, \hat{\nu}(t_{j-1}, \varphi_1), \hat{\nu}(t_j, \varphi_2)\}$ must evaluate to

B by stating that we cannot derive that φ is true (i. e., the case described by (B27)).

```
truthValue(F,S_i,b) :- until(F,G,H),
                       state(S_i),
                       state(S_j),
                       S_j > S_i,
                       S_j <= M,
                       finalState(M),
                       X{truthValue(G,S,t): state(S),
                              S >= S_i,
                              S < S_j;
                              truthValue(H,S_j,t);
                              truthValue(G,S,b): state(S),
                              S >= S_i,
                              S < S_j;
                              truthValue(H,S_j,b)}X,
                       X = S_j - S_i + 1,
                       not truthValue(F,S_i,t).          (B29)
```

Lastly, $\varphi = \varphi_1 \mathbf{U} \varphi_2$ is false if it is neither true nor both:

```
truthValue(F,S_i,f) :- until(F,_,_),
                       state(S_i),
                       not truthValue(F,S_i,t),
                       not truthValue(F,S_i,b).          (B30)
```

A formula $\varphi = \mathbf{G}\varphi_1$ evaluates to false in state t_i if φ_1 is false in any (following) state from t_i, i. e., φ evaluates to false if there exists at least one state t_j with $j \geq i$ in which φ_1 evaluates to false:

```
truthValue(F,S_i,f) :- globally(F,G),
                       state(S_i),
                       1{truthValue(G,S,f): state(S),
                              S >= S_i}.                 (B31)
```

The formula is true in t_i, if φ_1 evaluates to true in all following states:

```
truthValue(F,S_i,t) :- globally(F,G),
                       state(S_i),
                       X{truthValue(G,S,t): state(S),
                              S >= S_i}X,
                       finalState(M),
                       X = M - S_i.                 (B32)
```

Otherwise (i. e., if φ_1 evaluates at least once to both, and to true in all other cases), the formula evaluates to both:

```
truthValue(F,S_i,b) :- globally(F,_),
                       state(S_i),
                       not truthValue(F,S_i,t),
                       not truthValue(F,S_i,f).     (B33)
```

A formula $\varphi = \mathbf{F}\varphi_1$ evaluates to false in state t_i if φ_1 is false in all following states and including the current:

```
truthValue(F,S_i,f) :- eventually(F,G),
                       state(S_i),
                       X{truthValue(G,S,f): state(S),
                              S >= S_i}X,
                       finalState(M),
                       X = M - S_i.                 (B34)
```

$\varphi = \mathbf{F}\varphi_1$ evaluates to true in state t_i if there is a state t_j with $j \geq i$ in which φ_1 evaluates to true:

```
truthValue(F,S_i,t) :- eventually(F,G),
                       state(S_i),
                       state(S_j),
                       truthValue(G,S_j,t),
                       S_j >= S_i.                  (B35)
```

Otherwise, the formula evaluates to both:

```
truthValue(F,S_i,b) :- eventually(F,_),
                       state(S_i),
                       not truthValue(F,S_i,t),
                       not truthValue(F,S_i,f).       (B36)
```

Overall, we aim to derive a satisfied interpretation, i. e., the truth value of each formula must not be false in t_0:

```
:- truthValue(F,0,f), kbElement(F).                   (B37)
```

We are now ready to define the actual inconsistency measures $\mathcal{I}_d^{LTL}$ and $\mathcal{I}_c^{LTL}$. To determine whether any atom in a given state evaluates to B (i. e., whether a given state is in AffectedStates), we define:

```
affectedState(S) :- state(S),
                    truthValue(A,S,b),
                    atom(A).                          (D1)
```

This allows us to minimize |AffectedStates| by means of an optimization statement:

```
#minimize{S: affectedState(S)}.                       (D2)
```

Let $R_{\text{base}}(\mathcal{K})$ be the union of the rules defined by (B1)–(B37) wrt. a knowledge base $\mathcal{K}$[9], and let R_{d} be the union of rules (D1) and (D2). The optimal solution of the program $P_{\text{d}}(\mathcal{K}) = R_{\text{base}}(\mathcal{K}) \cup R_{\text{d}}$ now corresponds exactly to the value of $\mathcal{I}_d^{LTL}(\mathcal{K})$.

In order to compute $\mathcal{I}_c^{LTL}$, we do not require `affectedState/1`—instead, we make use of the `#count` aggregate in order to count the number of B evaluations in a given state:

```
numBInState(S,X) :- state(S),
                    #count{A: truthValue(A,S,b),
                              atom(A)} = X.           (C1)
```

[9]Note the rules yielded by (B2)–(B8) will only appear in $R_{\text{base}}(\mathcal{K})$ if the corresponding operators are part of the original $\mathcal{K}$.

To sum up the values of `numBInState/2`, we use a `#sum` aggregate:

$$\texttt{sumB(X) :- \#sum\{Y,S: numBInState(S,Y), state(S)\} = X.} \tag{C2}$$

Finally, we minimize the aforementioned sum:

$$\texttt{\#minimize\{X: sumB(X)\}.} \tag{C3}$$

Let R_c be the union of rules (C1)–(C3). The optimal answer set of the program $P_\mathrm{c}(\mathcal{K}) = R_\mathrm{base}(\mathcal{K}) \cup R_\mathrm{c}$ then represents the value of $\mathcal{I}_c^{LTL}(\mathcal{K})$.

We conclude this section with an example illustrating the concrete retrieval of an inconsistency measure value.

Example 25. *Consider the following knowledge base $\mathcal{K}_9$:*

$$\mathcal{K}_9 = \left\{ \underbrace{\overbrace{a}^{\varphi_{1,1}} \boldsymbol{U} \overbrace{b}^{\varphi_{1,2}}}_{\varphi_1}, \quad \underbrace{\boldsymbol{G} \overbrace{a}^{\varphi_{2,1}}}_{\varphi_2}, \quad \underbrace{\boldsymbol{X} \overbrace{\neg \overbrace{a}^{\varphi_{3,1,1}}}^{\varphi_{3,1}}}_{\varphi_3} \right\}$$

In this example, we construct $P_\mathrm{d}(\mathcal{K}_9)$ and $P_\mathrm{c}(\mathcal{K}_9)$ with $m = 10$. To achieve this, we first need to construct $R_\mathrm{base}(\mathcal{K}_9)$. We begin by representing the formulas in $\mathcal{K}_7$ according to (B1) and add them to $R_\mathrm{base}(\mathcal{K}_9)$:

$$\texttt{kbElement(}\varphi_1\texttt{).} \quad \texttt{kbElement(}\varphi_2\texttt{).} \quad \texttt{kbElement(}\varphi_3\texttt{).}$$

*The first formula in $\mathcal{K}_9$ (φ_1) contains a **U** operator, so we represent it in ASP as* `until(`φ_1`,`$\varphi_{1,1}$`,`$\varphi_{1,2}$`)` *(B6). As φ_2 contains a **G** operator, we represent it as* `globally(`φ_2`,`$\varphi_{2,1}$`)` *(B7). Lastly, φ_3 contains an **X** operator, so it is represented as* `next(`φ_3`,`$\varphi_{3,1}$`)` *(B5). Further, $\varphi_{3,1}$ consists of a negation, thus we need to add* `negation(`$\varphi_{3,1}$`,`$\varphi_{3,1,1}$`)` *(B4) to $R_\mathrm{base}(\mathcal{K}_9)$. To finally represent those subformulas that consist of individual atoms, we follow (B9):*

$$\begin{array}{ll} \texttt{formulaIsAtom}(\varphi_{1,1}, a). & \texttt{formulaIsAtom}(\varphi_{1,2}, b). \\ \texttt{formulaIsAtom}(\varphi_{2,1}, a). & \texttt{formulaIsAtom}(\varphi_{3,1,1}, a). \end{array}$$

To represent the atoms in the signature, i. e., $\mathsf{At}(\mathcal{K}_9) = \{a, b\}$, using (B10) we add:

$$\texttt{atom}(a). \quad \texttt{atom}(b).$$

The formulas in $\mathcal{K}_9$ are now represented as ASP rules. In order to model the number of states, we need to represent that the final state m is set to 10*. Thus, we add* `finalState(10)` *(B12).*

The remainder of the rules required to form $R_{\text{base}}(\mathcal{K}_9)$ are static, meaning that they do not need to be adapted to fit $\mathcal{K}_9$. Hence, we can add (B11) and (B13)–(B36) exactly as they were previously defined. Note that the rules referring to $\wedge$, $\vee$, and $\boldsymbol{F}$ (i. e., (B16)–(B18), (B19)–(B21), and (B33)–(B35), respectively) are not strictly necessary, as those operators do not appear in $\mathcal{K}_9$.

In order to ultimately form $P_{\text{d}}(\mathcal{K}_9)$ and $P_{\text{c}}(\mathcal{K}_9)$ we simply need to join $R_{\text{base}}(\mathcal{K}_9)$ with R_{d} and, respectively, R_{c}.

Two further remarks on reasoning with preference relations and on culpability measurement are in order.

Regarding measuring inconsistency with preferences, note that this can be added in a straightforward manner. Definition 8 provides two additional constraints that need to be considered. For this, a tagging is induced via a simple graph search following the stated inference rules. To calculate #openConflictingStates (see Definition 9), we do not even need to alter the proposed ASP encoding for $\mathcal{I}_d^{LTL}$ at all—we simply apply it to all formulas tagged $+\delta$ and disregard the formulas tagged $-\delta$. To calculate #exceptionStates (see Definition 10), we can compute the subtrahend by once again applying the aforementioned ASP encoding. As for the minuend, we finally need to slightly extend the previous ASP encoding in order to incorporate the two conditions proposed in Definition 8. Thus, we need to include predicates that represent which formulas tagged as $+\delta$ and $-\delta$, respectively. This additional information can then be used to extend the integrity constraint (B37) in such a way that it is limited exclusively to formulas tagged $+\delta$, and therefore model condition 1. from Definition 8. All that is now left to do is to model the second condition, i. e., that no formulas tagged $-\delta$ are unnecessarily unsatisfied.

Regarding culpability measurement, the presented approaches for computing the inconsistency measures can be extended in a straightforward way to obtain the corresponding Shapley values (i. e., simply by computing the values over all subsets of formulas, cf. Section 4.3). However, it would also be possible to leverage the previous results as heuristics for reducing the computational burden here. As an example, one only needs to iterate over all subsets of formulas that contain at least one atom which was assigned a truth value B—the marginal contribution towards the overall inconsistency will always be 0 in all other cases. In future works, we aim to investigate this in more detail, possibly also extending our results for Shapley value approximation.

7. Evaluation

We continue with an evaluation of our approach for time-sensitive inconsistency measurement. We briefly recall results on the computational complexity of our approach, which have already been presented in [9]. For this, let $\mathbb{L}$ denote the set of all LTL$_{\text{ff}}$ knowledge bases. We assume familiarity with basic concepts of computational complexity and basic complexity classes such as P and NP, see [49] for an introduction. Proofs can be found in the Appendix.

Following [11], we consider the following computational problems:

$\textsc{Exact}_{\mathcal{I}}$	**Input**:	$\mathcal{K} \in \mathbb{L}$, $x \in \mathbb{R}^{\infty}_{\geq 0}$
	Output:	TRUE iff $\mathcal{I}(\mathcal{K}) = x$
$\textsc{Upper}_{\mathcal{I}}$	**Input**:	$\mathcal{K} \in \mathbb{L}$, $x \in \mathbb{R}^{\infty}_{\geq 0}$
	Output:	TRUE iff $\mathcal{I}(\mathcal{K}) \leq x$
$\textsc{Lower}_{\mathcal{I}}$	**Input**:	$\mathcal{K} \in \mathbb{L}$, $x \in \mathbb{R}^{\infty}_{\geq 0} \setminus \{0\}$
	Output:	TRUE iff $\mathcal{I}(\mathcal{K}) \geq x$
$\textsc{Value}_{\mathcal{I}}$	**Input**:	$\mathcal{K} \in \mathbb{L}$
	Output:	The value of $\mathcal{I}(\mathcal{K})$

For $\textsc{Upper}_{\mathcal{I}}$, the same general non-deterministic algorithm as in Theorem 1 from [9] can be applied.

Theorem 1. *$\textsc{Upper}_{\mathcal{I}_d^{LTL}}$ and $\textsc{Upper}_{\mathcal{I}_c^{LTL}}$ are NP-complete.*

Using the insights from [11] we also get the following results for the other problems.

Corollary 1. *$\textsc{Lower}_{\mathcal{I}_d^{LTL}}$ and $\textsc{Lower}_{\mathcal{I}_c^{LTL}}$ are coNP-complete. $\textsc{Exact}_{\mathcal{I}_d^{LTL}}$ and $\textsc{Exact}_{\mathcal{I}_c^{LTL}}$ are in DP. $\textsc{Value}_{\mathcal{I}_d^{LTL}}$ and $\textsc{Value}_{\mathcal{I}_c^{LTL}}$ are in $\mathsf{FP}^{\mathsf{NP}[\log n]}$.*

So we see that in general, computing the concrete values is intractable. Therefore, to investigate the plausibility of applying our approach in practice, we conducted runtime experiments with real-life data sets. In the following, we present our evaluation results. We begin by briefly introducing our data set.

7.1. Description of Data Sets

We conducted a series of runtime experiments for the ASP-based implementations of $\mathcal{I}_d^{LTL}$ and $\mathcal{I}_c^{LTL}$ (see Section 6) using real-life data sets of the Business Process Intelligence Challenge (BPIC)[10]. The BPIC is an international scientific challenge series and provides real-life process data from various domains such as healthcare, government or industry. The process data provided via the BPIC is offered in the form of so-called event logs, which are essentially (multi)sets of activity sequences (i. e., [customer] cases). In our setting, these correspond to linear sequences of time-points, where at every point in time a certain activity may occur. Importantly, various tools exist that allow to *mine* LTL rule sets from this case data (essentially, temporal rules that hold over the sequences in the data sets can be mined). Here, we applied the state-of-the-art tool MINERful [50, 51] to obtain LTL rule sets from the BPIC data sets. As mining parameters, we selected standard mining parameters as suggested in [4], namely a support factor of 75% (minimum number of cases a rule has to be fulfilled in), as well as confidence and interest factors of 12.5% (support scaled by the ratio of cases in which the activation occurs, resp. support scaled by the ratio of cases both the activation and reaction occur). All obtained LTL rule sets can be found online[11].

The data sets of the BPIC are ideal candidates for our evaluation, as they allow us to obtain LTL rule sets from event logs of industrial complexity. For our evaluation, we used the BPIC data sets of the last five available challenges (2016–2020), which we briefly outline in the following:

- **BPIC 2016.** This data set contains process data of a healthcare process for treating Sepsis and contains around 1.000 cases.
- **BPIC 2017.** This data set contains process data of a loan application process of a vendor in the financial industry and contains around 30.000 cases.
- **BPIC 2018.** This data set contains process data of government fund distribution process and contains around 43.000 cases.

[10]`https://icpmconference.org/2020/bpi-challenge/`

[11]`https://github.com/aig-hagen/inconsistency-measurement-LTL/tree/master/data`

- **BPIC 2019.** This data set contains process data of a purchase process a of a large multinational company and contains around 250.000 cases.
- **BPIC 2020.** This data set contains process data of a travel reimbursement process of a university and contains around 32.000 cases.

The 2020 data set is divided into five sub logs, marked accordingly. Note that we also included the data set from 2012, as this is an earlier version of the 2017 data set.

We then applied our presented implementation to compute the inconsistency measures for the obtained LTL rule sets (see above).

In Table 3, we show an overview of the analyzed data sets, including the domain, the number of (customer) cases contained in the original data set, the average sequence length in the original data set, as well as the number of distinct atoms in the respective rule bases. As can be seen, the considered data sets reflect case data of industrial complexity with up to 250.000 cases. Table 3 also indicates the size of these obtained LTL rule sets. As shown, the sizes of the considered rule bases ranged from around 60–600 rules.

	Data set	Domain	# of cases	∅ case length	# of atoms	# of rules
1	BPIC2012	Financial Industry	13 087	20.03	21	275
2	BPIC2016	Healthcare	1 050	14.48	11	204
3	BPIC2017	Financial Industry	31 509	38.15	20	607
4	BPIC2018	Government	43 809	47.39	17	449
5	BPIC2019	Industry	251 734	6.33	8	65
6	BPIC2020_1	Government	6 886	5.37	6	58
7	BPIC2020_2	-	7 065	11.18	14	295
8	BPIC2020_3	-	2 099	12.25	18	374
9	BPIC2020_4	-	6 449	8.69	12	194
10	BPIC2020_5	-	10 500	5.34	7	66

Table 3: Overview of the considered data sets including industrial domain, number of cases, average case length, number of atoms, and size of the rule-base obtained via the described mining tool.

We now continue to present our experimental results.

7.2. Experimental Results

The experiments were run on a computer with 32 GB RAM and an AMD Ryzen 7 PRO 5850U CPU which has a basic clock frequency of 1.9 GHz. The approach was implemented in C++, and Clingo 5.4.0 is the ASP solver we used. As Table 3 shows, the average case length of the different data sets lies between 5.34 (BPIC2020_5) and 47.39 (BPIC2018). We therefore decided to measure the runtime wrt. both $\mathcal{I}_d^{LTL}$ and $\mathcal{I}_c^{LTL}$ with $m \in \{10, 20, 30, 40, 50\}$. Moreover, we set a timeout to 900 seconds (15 minutes). Figure 1 shows our experimental results.

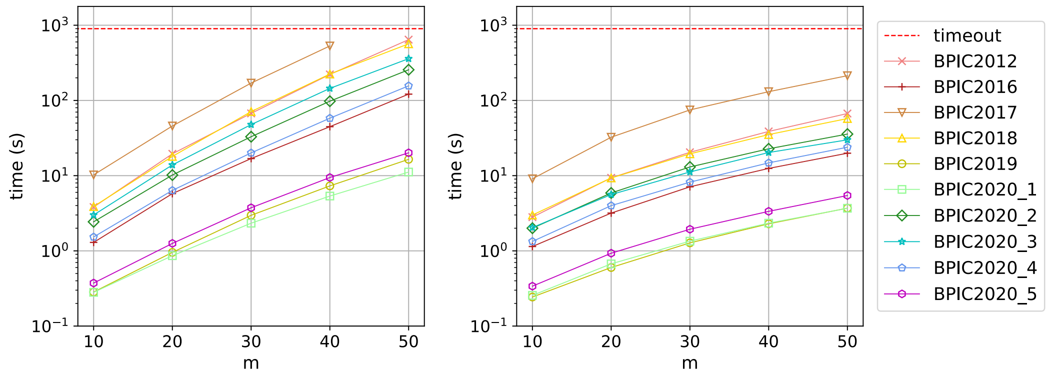

Figure 1: Results of runtime experiments wrt. $\mathcal{I}_c^{LTL}$ (left) and $\mathcal{I}_d^{LTL}$ (right). A timeout was set to 900 seconds.

In general, the computation of both inconsistency measures was feasible for the considered real-life data sets. For a smaller number of states (e. g., 10), the values could be computed in a few seconds. For some larger configurations (e. g., m=50), this could take up to 10 minutes. We also observe that only in one case, a timeout occurred. To be precise, this happened for the BPIC2017 instance with $m = 50$ for $\mathcal{I}_c^{LTL}$. However, the average case length for this instance is 38.15, and for $m = 40$, $\mathcal{I}_c^{LTL}$ could be computed in less than 10 minutes.

As can be seen, the runtime scales in relation to the number of states (which is to be expected, as more states also mean more possible solutions for the ASP solver). In this sense, the configuration of the number of states to be considered is crucial for the performance. Here, we argue that the average case length of the respective data set may serve as a good indication for how to select this parameter (cf. Table 3). Here, a main observation is that the inconsistency values for each data set wrt. each respective average case length could be computed well within the time limit (over all data sets, the average case length was around 16.9, so assuming a parameter of $m = 20$, note that all values could be computed in well under 100 seconds for this setting).

Another observation is that $\mathcal{I}_d^{LTL}$ seems to be computed faster than $\mathcal{I}_c^{LTL}$ wrt. a given data set and number of states. Again, this is rather unsurprising, as the search space of possible solutions is generally smaller regarding $\mathcal{I}_d^{LTL}$

than regarding $\mathcal{I}_c^{LTL}$. Moreover, the runtime measurements for both inconsistency measures show a similar tendency in terms of the different data sets (e. g., calculating the values for BPIC2017 takes longer than for any other data set for both $\mathcal{I}_d^{LTL}$ and $\mathcal{I}_c^{LTL}$).

Note that as the data sets are unrelated, no further comparison can be made as to how the performance scales with the number of rules, or the number of atoms. In future work, we aim to conduct further experiments to investigate this further. It may also be interesting for future works to investigate how the concrete distribution of how often the different temporal operators occur impacts performance.

7.3. Outlook: Approximation

The above experimental results indicate general feasibility for the considered lengths of sequences (as mentioned based on observed real-life trace lengths). In general, the topic of approximating inconsistency values seems however interesting for our use-case, as the length could be identified to be a factor to the runtime, e. g., regarding even longer trace lengths. As an outlook, we present an algorithm for approximating whether m states will always be affected, for any m. To clarify, if less than the considered number of time points are affected, a concrete value of $\mathcal{I}_d^{LTL}$ could still increase for larger m, e. g., in case of loops. However, we can approximate whether it is impossible for m time points to be affected in general. Vice versa, it suffices to consider the maximal depth over all formulas as m to approximate whether in fact all time points will always be affected for any m (in which case $\mathcal{I}_d^{LTL}$ is in fact always equal to the considered number of states).

For our approach, an important observation to recall (cf. also Section 3) is that via Proposition 3, m can be set to the depth of the conjunction of all formulas in a knowledge base $\mathcal{K}$. This ensures that an interpretation exists. It directly follows that if an inconsistency affects $n < m$ states, it can never hold that all states will be affected as we quantify inconsistency by seeking a minimal number of assignments to the truth value B (and we have already found a minimal one here). Consequently, setting m to the depth suffices in this case already. For the case that m states are affected, Algorithm 1 can be used to efficiently compute whether this will be the case for any m, if m is increased.

Example 26. *Recall* $\mathcal{K}_1 = \{\boldsymbol{X}a, \boldsymbol{X}\neg a\}$ *and* $\mathcal{K}_2 = \{\boldsymbol{G}a,\ \boldsymbol{G}\neg a\}$*. For both knowledge bases,* $d = 1$*. So initially we assume* $m = 1$*, i.e., states* t_0, t_1*.*

Algorithm 1: Deciding whether $\mathcal{I}_d^{LTL}(\mathcal{K})$ will be $m(+1)$, for any m

```
   Input   : Knowledge base $\mathcal{K}$
   Output: TRUE if $\mathcal{I}_d^{LTL}(\mathcal{K}) = m + 1$ for any $m$, FALSE otherwise
 1 $depth \leftarrow d(\bigcup \mathcal{K})$
 2 $m \leftarrow depth$
 3 $\#affectedStatesInitial \leftarrow \mathcal{I}_d^{LTL}(\mathcal{K})$// Assuming m
 4 if $\#affectedStatesInitial \leq m$ then
 5     return FALSE
 6 $m' \leftarrow m + 1$
 7 $\#affectedStates' \leftarrow \mathcal{I}_d^{LTL}(\mathcal{K})$// Assuming m'
 8 if $\#affectedStates' = m' + 1$ then
 9     return TRUE
10 else
11     return FALSE
```

Via line 3, we see that $\mathcal{I}_d^{LTL}(\mathcal{K}_1) = 1$ and $\mathcal{I}_d^{LTL}(\mathcal{K}_2) = 2 = m + 1$. So for $\mathcal{K}_1$, we correctly return false. This is because not all time points $t_0, ..., t_m$ are affected. Note also that this value cannot increase for $\mathcal{K}_1$, as we have already identified a minimal interpretation and this would remain a valid minimal interpretation even if m is increased. For $\mathcal{K}_2$, we again compute $\mathcal{I}_d^{LTL}$ assuming $m' = m + 1 = 2$ (via line 7). We again see that $\mathcal{I}_c^{LTL}(\mathcal{K}_2) = 3 = m' + 1$, and can correctly return true. Importantly, we know via Proposition 3 that an interpretation already exists (and via $\mathcal{I}_d^{LTL}$ we have selected one with a minimal number of assignments to B). The condition in line 8 is needed to discriminate cases that affect exactly $m + 1$ time points, but do not grow, if m grows. For example, if the depth of a knowledge base is 1 and the inconsistency affects initially exactly 1 state + t_0, it could still be that when increasing m, $\mathcal{I}_d^{LTL}$ will remain 1+1 (in which case it does not hold that m states will always be affected).

We continue to discuss the *correctness* of Algorithm 1 (cf. Appendix A). Here, we consider an algorithm to be correct if it satisfies the properties of *soundness* and *completeness*, which we define as follows:

- Given a problem P and an algorithm that tries to solve this problem, an algorithm is sound if the solution returned by the algorithm is correct.

- Given a problem P and an algorithm that tries to solve this problem, an algorithm is complete if it returns a solution if one exists, or reports failure, if no solution exists.

Theorem 2. *For a knowledge base $\mathcal{K}$, Algorithm 1 is sound for the problem of deciding whether $\mathcal{I}_d^{LTL}(\mathcal{K})$ will be $m+1$, for any $m > 0$.*

Theorem 3. *For a knowledge base $\mathcal{K}$, Algorithm 1 is complete for the problem of deciding whether $\mathcal{I}_d^{LTL}(\mathcal{K})$ will be $m+1$, for any $m > 0$.*

To gain an overview of the potential interest for Algorithm 1 in practical settings, we computed the times for running Algorithm 1 for the considered real-life data-sets, shown in Table 4. As can be seen, the algorithm seems very useful here, as the max. depth is very low (which is intuitive, given the predefined template structure of Declare), and thus computation was very fast in all cases. In result, it seems plausible to apply our approach to reason whether in fact all time points will always be affected, without having to consider large values for m (which may be computationally hard). One could also easily extend our algorithm with a bound parameter b to assess whether $m-b$ states will always be affected. Also, note that it may be useful to extend the algorithm to reason about whether $\mathcal{I}_d^{LTL}$ will never increase (cf. the discussion for $\mathcal{K}_1$ in Example 26).

	Data set	Max. depth	Runtime (s)
1	BPIC2012	3	0.583
2	BPIC2016	3	0.304
3	BPIC2017	3	1.593
4	BPIC2018	3	0.686
5	BPIC2019	3	0.095
6	BPIC2020_1	3	0.110
7	BPIC2020_2	3	0.605
8	BPIC2020_3	3	0.658
9	BPIC2020_4	3	0.428
10	BPIC2020_5	3	0.145

Table 4: Runtime results when m is set to $d(\mathcal{K})$.

The presented Algorithm 1 focused on $\mathcal{I}_d^{LTL}$, which is built on the number of affected states, i.e., dependent of m. For measures such as $\mathcal{I}_c^{LTL}$, it is not possible to easily define a function that maps the value of the inconsistency measure relative to m. However, given that the length of the time sequence seems to play an important role for computation, approximation techniques seem promising in general and should be considered in future work.

8. Conclusion

In this work, we have presented an approach for measuring the severity of inconsistencies in declarative process specifications, in particular those based on linear temporal logic. In this regard, we introduced a paraconsistent semantics for $\mathrm{LTL}_{\mathrm{ff}}$ and developed inconsistency measures, as well as element-based measures. Here, our approach extends recent works [3, 4, 5] on the identification of inconsistent sets in declarative process specifications by allowing a look "into" those sets. Also, we have shown how our results apply to higher-level modeling languages such as Declare, allowing also a wider range of stakeholders to access our results, with the underlying complexity of LTL hidden from the user.

The presented work also extends our previous results in [9], by presenting novel algorithmic approaches for computing the devised measures. Here, we show that computation is feasible for real-life data sets of the BPI challenge, allowing us to implement our results in applied settings. We aim to conduct further experiments in future work.

As a further addition, we have extended our framework by allowing to model preferences amongst formulas, which allows to define default rules and exceptions in a flexible manner. Being able to express concepts like exceptions seems to be an important aspect for allowing companies to handle declarative specifications properly, in particular, in use-cases including knowledge changes over time, uncertainty, or other data-perspectives of declarative process specifications such as resources [52, 45]. Here, it seems useful to also be able to lift results for inconsistency measurement to these use-cases.

On a more general note, LTL research has traditionally been focused on answering certain questions in a binary manner, e. g., "are certain formulas satisfied?", or "is the set of formulas inconsistent?". While this work is the first to investigate the notion of a *degree* of inconsistency in LTL, we see some other works that try to answer other LTL-related research questions in a more quantitative fashion. For example, in [53, 54], the authors investigate the notion of a *satisfaction degree*, which, in short, is a degree between [0,1] indicating "how much" an individual LTL formula is satisfied by an interpretation. While this is a different problem than addressed in this paper, it seems that the idea of quantifying certain LTL-related notions may be promising for future work.

References

[1] A. Pnueli, The temporal logic of programs, in: 18th Symposium on Foundations of Computer Science, Rhode Island, IEEE Computer Society, 1977, pp. 46–57.

[2] A. Cecconi, G. De Giacomo, C. Di Ciccio, F. M. Maggi, J. Mendling, A temporal logic-based measurement framework for process mining, in: Proceedings of the 2nd ICPM, IEEE, 2020, pp. 113–120.

[3] M. Roveri, C. Di Ciccio, C. Di Francescomarino, C. Ghidini, Computing unsatisfiable cores for LTLf specifications (Preprint), arXiv, 2022.

[4] C. Di Ciccio, F. M. Maggi, M. Montali, J. Mendling, Resolving inconsistencies and redundancies in declarative process models, Inf. Systems 64 (2017) 425–446.

[5] C. Corea, S. Nagel, J. Mendling, P. Delfmann, Interactive and minimal repair of declarative process models, in: BPM Forum, Rome, Springer, 2021, pp. 3–19.

[6] J. Grant, M. V. Martinez, Measuring Inc. in Information, College Pub., 2018.

[7] M. Thimm, Inconsistency measurement, in: Proceedings of the 13th International Conference on Scalable Uncertainty Management, Compiègne, Springer, 2019.

[8] C. Corea, J. Grant, M. Thimm, Measuring inconsistency in declarative process specifications, in: International Conference on Business Process Management, Springer, 2022, pp. 289–306.

[9] C. Corea, J. Grant, M. Thimm, Measuring inconsistency in declarative process specifications, in: Proceedings of the 20th International Conference on Business Process Management (BPM 2022), Münster, Germany,, Vol. 13420 of Lecture Notes in Computer Science, Springer, 2022, pp. 289–306.

[10] A. Hunter, S. Konieczny, et al., Shapley inconsistency values., KR 6 (2006) 249–259.

[11] M. Thimm, J. P. Wallner, On the complexity of inc. meas., AI 275 (2019) 411–456.

[12] J. Grant, A. Hunter, Measuring consistency gain and inf. loss in stepwise inc. resolution, in: Proc. of the 11th ECSQARU, Belfast, Springer, 2011, pp. 362–373.

[13] G. Priest, Logic of Paradox, Journal of Phil. Logic 8 (1979) 219–241.

[14] A. Hunter, S. Konieczny, Measuring inconsistency through minimal inconsistent sets., KR 8 (2008) 358–366.

[15] V. Fionda, G. Greco, The compl. of LTL on finite traces: Hard and easy fragments, in: Proc. of the 30th AAAI Conference on AI, Phoenix, AAAI, 2016, pp. 971–977.

[16] G. De Giacomo, R. De Masellis, M. Montali, Reasoning on ltl on finite traces: Insensitivity to infiniteness, in: 28th AAAI Conference on AI, 2014.

[17] G. D. Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: Proceedings of the 23rd IJCAI, Beijing, AAAI, 2013, pp. 854–860.

[18] D. Solomakhin, M. Montali, S. Tessaris, R. D. Masellis, Verification of artifact-centric systems, in: Proceedings of the 11th ICSOC, Springer, 2013, pp. 252–266.

[19] T. Hildebrandt, R. R. Mukkamala, T. Slaats, F. Zanitti, Contracts for cross-organizational workflows as timed dynamic condition response graphs, The Journal of Logic and Algebraic Programming 82 (5-7) (2013) 164–185.

[20] I. Pill, T. Quaritsch, Behavioral diagnosis of ltl specifications at operator level, in: 23rd International Joint Conference on Artificial Intelligence, Citeseer, 2013.

[21] F. Hantry, M.-S. Hacid, Handling conflicts in depth-first search for ltl tableau to debug compliance based languages, arXiv preprint arXiv:1109.2656 (2011).

[22] V. Schuppan, Towards a notion of unsatisfiable and unrealizable cores for ltl, Science of Computer Programming 77 (7-8) (2012) 908–939.

[23] F. M. Maggi, M. Westergaard, M. Montali, W. M. van der Aalst, Runtime verification of ltl-based declarative process models, in: Proceedings of the 2nd RV, San Francisco, Springer, 2011, pp. 131–146.

[24] J. Grant, Measuring inconsistency in some branching time logics, Journal of Applied Non-Classical Logics 31 (2021) 85–107.

[25] M. Y. Vardi, Branching vs. linear time: Final showdown, in: Proceedings of the 7th TACAS, Italy, Springer, 2001, pp. 1–22.

[26] N. Kamide, H. Wansing, Combining linear-time temporal logic with constructiveness and paraconsistency, Journal of Applied Logic 8 (2010) 33–61.

[27] N. Kamide, H. Wansing, A paraconsistent linear-time temporal logic, Fundamenta Informaticae 106 (2011) 1–23.

[28] D. Nute, Defeasible logic, in: International Conference on Applications of Prolog, Springer, 2001, pp. 151–169.

[29] A. Chafik, F. Cheikh-Alili, J.-F. Condotta, I. Varzinczak, Defeasible linear temporal logic, Journal of Applied Non-Classical Logics (2023) 1–51.

[30] G. Governatori, P. Terenziani, Temporal extensions to defeasible logic, in: Australasian Joint Conference on Artificial Intelligence, Springer, 2007, pp. 476–485.

[31] A. Alman, F. M. Maggi, M. Montali, R. Peñaloza, Probabilistic declarative process mining, Information Systems 109 (2022) 102033.

[32] E. Bellodi, F. Riguzzi, E. Lamma, Probabilistic declarative process mining, in: International Conference on Knowledge Science, Engineering and Management, Springer, 2010, pp. 292–303.

[33] G. Antoniou, D. Billington, G. Governatori, M. J. Maher, Representation results for defeasible logic, ACM Transactions on Computational Logic (TOCL) 2 (2) (2001) 255–287.

[34] K. Mu, K. Wang, L. Wen, Approaches to measuring inconsistency for stratified knowledge bases, International Journal of Approximate Reasoning 55 (2) (2014) 529–556.

[35] I. Kuhlmann, M. Thimm, Algorithms for inconsistency measurement using answer set programming, in: Proceedings of the 19th International Workshop on Non-Monotonic Reasoning (NMR), 2021, pp. 159–168.

[36] F. Chiariello, F. M. Maggi, F. Patrizi, ASP-based declarative process mining, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2022, pp. 5539–5547.

[37] K. Heljanko, I. Niemelä, Bounded ltl model checking with stable models, Theory and Practice of Logic Programming 3 (4-5) (2003) 519–550.

[38] M. Thimm, On the expressivity of inconsistency measures, Artificial Intelligence 234 (2016) 120–151.

[39] N. Potyka, Measuring disagreement among knowledge bases, in: International Conference on Scalable Uncertainty Management, Springer, 2018, pp. 212–227.

[40] P. Felli, A. Gianola, M. Montali, A. Rivkin, S. Winkler, Conformance checking with uncertainty via smt, in: International Conference on Business Process Management, Springer, 2022, pp. 199–216.

[41] M. J. Maher, A model-theoretic semantics for defeasible logic, arXiv preprint cs/0207086 (2002).

[42] J. S. Ribeiro, M. Thimm, Consolidation via tacit culpability measures: Between explicit and implicit degrees of culpability., KR 2021 (2021) 529–538.

[43] L. S. Shapley, A value for n-person games, in: Contributions to the Theory of Games II, Annals of Mathematics Studies, vol. 28, Princeton University Press, 1953, pp. 307–317.

[44] M. Pesic, H. Schonenberg, W. M. Van der Aalst, Declare: Full support for loosely-structured processes, in: 11th IEEE EDOC, Annapolis, IEEE, 2007, pp. 287–287.

[45] A. Burattin, F. M. Maggi, A. Sperduti, Conformance checking based on multi-perspective declarative process models, Expert systems with applications 65 (2016) 194–211.

[46] I. Kuhlmann, M. Thimm, An algorithm for the contension inconsistency measure using reductions to answer set programming, in: Proceedings of the 14th International Conference on Scalable Uncertainty Management (SUM'20), Springer, 2020, pp. 289–296.

[47] I. Kuhlmann, A. Gessler, V. Laszlo, M. Thimm, A comparison of ASP-based and SAT-based algorithms for the contension inconsistency measure, in: Proceedings of the 15th International Conference on Scalable Uncertainty Management (SUM'22), Springer, 2022.

[48] I. Kuhlmann, A. Gessler, V. Laszlo, M. Thimm, Comparison of SAT-based and ASP-based algorithms for inconsistency measurement, arXiv preprint arXiv:2304.14832 (2023).

[49] C. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.

[50] C. Di Ciccio, M. H. Schouten, M. de Leoni, J. Mendling, Declarative process discovery with minerful in prom., 2015, pp. 60–64.

[51] A. Alman, C. Di Ciccio, D. Haas, F. M. Maggi, A. Nolte, Rule mining with rum, in: Int. Conference on Process Mining, 2020, pp. 121–128.

[52] F. M. Maggi, M. Montali, R. Peñaloza, Temporal logics over finite traces with uncertainty, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34, 2020, pp. 10218–10225.

[53] H.-X. Shi, A quantitative approach for linear temporal logic, in: Quantitative Logic and Soft Computing 2016, Springer, 2017, pp. 49–57.

[54] R. Dimitrova, M. Ghasemi, U. Topcu, Maximum realizability for linear temporal logic specifications, in: International Symposium on Automated Technology for Verification and Analysis, Springer, 2018, pp. 458–475.

[55] M. Thimm, On the evaluation of inconsistency measures, in: J. Grant, M. V. Martinez (Eds.), Measuring Inconsistency in Information, College Publications, 2018.

[56] G. De Bona, A. Hunter, Localizing iceberg dependencies, Artificial Intelligence 246 (2017) 118–151.

[57] A. Hunter, S. Konieczny, On the measure of conflicts: Shapley inconsistency values, Artificial Intelligence 174 (14) (2010) 1007–1026.

[58] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Communications of the ACM 54 (12) (2011) 92–103.

[59] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Answer set solving in practice, Synthesis lectures on artificial intelligence and machine learning 6 (3) (2012) 1–238.

[60] V. Lifschitz, Answer set programming, Springer Berlin, 2019.

[61] R. Reiter, A logic for default reasoning (1980).

Acknowledgements

We would like to thank the reviewers for their very helpful comments in improving this manuscript. The research reported here was supported by the Deutsche Forschungsgemeinschaft (grant DE 1983/9-2).

Appendix A: Proofs for Technical Results

Proposition 2. *For every (two-valued) LTL$_{ff}$ interpretation $\hat{\omega}$ and LTL$_{ff}$ formula ϕ, $\hat{\omega} \models \phi$ if and only if $\hat{\omega} \models^3 \phi$.*

Proof. Let $\hat{\omega}$ be any two-valued LTL$_{ff}$ interpretation. First observe that using three-valued semantics, $\hat{\omega}(t_i, \phi) \neq$ B for every ϕ (this can be easily verified by structural induction since no atom has the value B). We prove now the more general statement that $\hat{\omega}, t_i \models \phi$ if and only if $\hat{\omega}(t_i, \phi) = 1$ for any t_i by induction on the structure of ϕ:

- $\phi = a$ for $a \in \mathsf{At}$: From $\hat{\omega}, t_i \models \phi$ it follows $\hat{\omega}(t_i, a) = 1$ via classical semantics, which is equivalent to $\hat{\omega}(t_i, a) = 1$ via 3-valued semantics.
- $\phi = \neg\psi$: We have $\hat{\omega}, t_i \models \neg\psi$ iff $\hat{\omega}, t_i \not\models \psi$. By the induction hypothesis, this is equivalent to $\hat{\omega}(t_i, \psi) = 0$. This is equivalent to $\hat{\omega}(t_1, \neg\psi) = 1$.

- $\phi = \phi_1 \wedge \phi_2$: We have that $\hat{\omega}, t_i \models \phi_1 \wedge \phi_2$ is equivalent to $\hat{\omega}, t_i \models \phi_1$ and $\hat{\omega}, t_i \models \phi_2$. By the induction hypothesis, this is equivalent to $\hat{\omega}(t_i, \phi_1) = 1$ and $\hat{\omega}(t_i, \phi_2) = 1$. This is equivalent to $\hat{\omega}(t_i, \phi) = 1$.

- $\phi = \phi_1 \vee \phi_2$: We have that $\hat{\omega}, t_i \models \phi_1 \vee \phi_2$ is equivalent to $\hat{\omega}, t_i \models \phi_1$ or $\hat{\omega}, t_i \models \phi_2$. Without loss of generality assume $\hat{\omega}, t_i \models \phi_1$. By the induction hypothesis, this is equivalent to $\hat{\omega}(t_i, \phi_1) = 1$ which implies $\hat{\omega}(t_i, \phi) = 1$. This other direction is analogous.

- $\phi = \mathbf{X}\psi$: Observe first that for $i \geq m$ both $\hat{\omega}, t_i \not\models \phi$ and $\hat{\omega}(t_i, \phi) = 0$ hold. So assume that $i < m$. Then we have $\hat{\omega}, t_i \models \mathbf{X}\psi$ equivalent to $\hat{\omega}, t_{i+1} \models \psi$. By the induction hypothesis, this is equivalent to $\hat{\omega}(t_{i+1}, \psi) = 1$ and $\hat{\omega}(t_i, \mathbf{X}\psi) = 1$.

- $\phi = \varphi_1 \mathbf{U} \varphi_2$: Assume that $\hat{\omega}, t_i \models \varphi_1 \mathbf{U} \varphi_2$. Then there is some $j \in \{i + 1, \ldots, m\}$ with $\hat{\omega}, t_j \models \phi_2$ and $\hat{\omega}, t_k \models \phi_1$ for all $k \in \{i, \ldots, j-1\}$. By the induction hypothesis, this amounts to $\hat{\omega}(t_j, \phi_2) = 1$ and $\hat{\omega}(t_k, \phi_1) = 1$ for all $k \in \{i, \ldots, j-1\}$. From this follows $\hat{\omega}(t_i, \phi_1 \mathbf{U} \phi_2) = 1$. The other direction is analogous. □

Proposition 3. *For every LTL$_{ff}$ formula ϕ with $d(\phi) \leq m$ there is a $\hat{\nu}$ with $\hat{\nu} \models^3 \phi$.*

Proof. Let $\hat{\nu}_b$ be the three-valued interpretation defined via $\hat{\nu}_b(t_i, a) = \mathrm{B}$ for all t_i and a. We prove the more general statement that $\hat{\nu}_b(t_i, \phi) = \mathrm{B}$ for any t_i and $d(\phi) \leq m - i$ by induction on the structure of ϕ:

- $\phi = a$ for $a \in \mathsf{At}$: $\hat{\nu}_b(t_i, a) = \mathrm{B}$ holds by definition.

- $\phi = \neg\psi$: $d(\phi) \leq m - i$ implies that $d(\psi) \leq m - i$ as well. By the induction hypothesis, $\hat{\nu}_b(t_i, \psi) = \mathrm{B}$ and therefore $\hat{\nu}_b(t_i, \neg\psi) = \mathrm{B}$ as well.

- $\phi = \phi_1 \wedge \phi_2$: $d(\phi) \leq m - i$ implies that $d(\phi_1) \leq m - i$ and $d(\phi_2) \leq m - i$ as well. By the induction hypothesis, $\hat{\nu}_b(t_i, \phi_1) = \hat{\nu}_b(t_i, \phi_2) = \mathrm{B}$ and therefore $\hat{\nu}_b(t_i, \phi_1 \wedge \phi_2) = \mathrm{B}$.

- $\phi = \phi_1 \vee \phi_2$: $d(\phi) \leq m - i$ implies that $d(\phi_1) \leq m - i$ and $d(\phi_2) \leq m - i$ as well. By the induction hypothesis, $\hat{\nu}_b(t_i, \phi_1) = \hat{\nu}_b(t_i, \phi_2) = \mathrm{B}$ and therefore $\hat{\nu}_b(t_i, \phi_1 \vee \phi_2) = \mathrm{B}$.

- $\phi = \mathbf{X}\psi$: $d(\phi) \leq m-i$ implies that $d(\psi) \leq m-(i+1)$. By the induction hypothesis $\hat{\nu}_b(t_{i+1}, \psi) = \mathrm{B}$ which directly gives $\hat{\nu}_b(t_i, \mathbf{X}\psi) = \mathrm{B}$.

- $\phi = \varphi_1 \mathbf{U} \varphi_2$: $d(\phi) \leq m-i$ implies that $d(\phi_1) \leq m-(i+1)$ and $d(\phi_2) \leq m-(i+1)$ as well. So, by the induction hypothesis $\hat{\nu}_b(t_{i+1}, \phi_2) = \mathrm{B}$ and therefore $\hat{\nu}_b(t_i, \varphi_1 \mathbf{U} \varphi_2) = \mathrm{B}$. □

Proposition 4. *$\mathcal{C}(\mathcal{I}_d^{LTL}, m) = m + 2$ and $\mathcal{C}(\mathcal{I}_c^{LTL}, m) = \infty$.*

Proof. For any interpretation $\hat{\nu}$, any number of the states $t_0, t_1, \ldots, t_m$, including none, may be affected by the inconsistency. This results in $\mathcal{C}(\mathcal{I}_d^{LTL}, m)$ $= m + 2$. $\mathcal{C}(\mathcal{I}_c^{LTL}, m) = \infty$ follows from the fact that already for a single state $\mathcal{I}_c^{LTL}$, m may take any non-negative integer value. For example, for $\mathcal{K}_i = \{a_1 \wedge \neg a_1, ..., a_i \wedge \neg a_i\}$. $\mathcal{I}_c^{LTL}(\mathcal{K}_i) = i$ and $\lim_{i \to \infty} \mathcal{I}_c^{LTL}(\mathcal{K}_i) = \infty$. So $\mathcal{C}(\mathcal{I}_c^{LTL}, n) = \infty$ for any number of states (given there is at least one state). □

Proposition 5. *The compliance of the inconsistency measures $\mathcal{I}_d$, $\mathcal{I}_{\mathsf{MI}}$, $\mathcal{I}_p$, $\mathcal{I}_r$, $\mathcal{I}_c$, $\mathcal{I}_{at}$, $\mathcal{I}_d^{LTL}$ and $\mathcal{I}_c^{LTL}$ with the postulates* CO, MO, IN, DO *and* TS *is as shown in Table 1.*

Proof. The proofs for $\mathcal{I}_d, \mathcal{I}_{\mathsf{MI}}, \mathcal{I}_p, \mathcal{I}_r, \mathcal{I}_c, \mathcal{I}_{at}$ can be found in [55].[12] In the process we corrected one result in [55], based on an earlier error. Actually, $\mathcal{I}_c$ does not satisfy IN because of iceberg inconsistencies (see [56]). For example, consider a knowledge base $\mathcal{K}_c = \{a \wedge \neg a \wedge b, \neg b\}$. Then $\neg b$ is free, but $\mathcal{I}_c(\mathcal{K}_c) = 2$ and $\mathcal{I}_c(\mathcal{K}_c \setminus \{\neg b\}) = 1$. Also, for those measures TS follows from Example 2.

We now consider the remaining measures $\mathcal{I}_d^{LTL}$ and $\mathcal{I}_c^{LTL}$ in turn. Recall that the definition of consistency for LTL requires the existence of a 2-valued interpretation $\hat{\nu}$ such that $\hat{\nu} \models \mathcal{K}$. Using such an interpretation we obtain $|\mathsf{AffectedStates}(\hat{\nu})| = |\mathsf{ConflictBase}(\hat{\nu})| = 0$. If $\mathcal{K}$ is inconsistent, B is assigned to at least one proposition, so these values are positive.

We now start with $\mathcal{I}_d^{LTL}$. For this, let $\mathcal{K}, \mathcal{K}'$ be knowledge bases and α, β be two formulas of $\mathrm{LTL}_{\mathrm{ff}}$. CO follows directly from the definition of consistency. For MO, if $\mathcal{K}$ is consistent, then $\mathcal{K} \cup \mathcal{K}'$ is either consistent or inconsistent. In both cases $\mathsf{AffectedStates}(\mathcal{K} \cup \mathcal{K}') \geq \mathsf{AffectedStates}(\mathcal{K})$.

[12]$\mathcal{I}_r$ is equivalent to $\mathcal{I}_{dalal}^{hit}$ from that work. Also, the proofs for $\mathcal{I}_{at}$ are analogous to $\mathcal{I}_{mv}$.

If $\mathcal{K}$ is inconsistent, so is $\mathcal{K} \cup \mathcal{K}'$ and again $\mathcal{I}_d^{LTL}(\mathcal{K}) \leq \mathcal{I}_d^{LTL}(\mathcal{K} \cup \mathcal{K}')$. For IN, observe that for any free formula α, there exists an interpretation that maps α to 1; thus, free formulas cannot affect the size of $\mathsf{Affected}$-States. In turn, if α is a free formula, for any interpretation $\hat{\nu}$ that assigns B to a minimal number of propositions we have that $|\mathsf{AffectedStates}(\hat{\nu})|$ is the same for $\mathcal{K}$ and $(\mathcal{K} \setminus \alpha)$. For DO, observe from [57] that if $\alpha \models \beta$, then $\{\hat{\nu} \mid \hat{\nu} \models^3 (\mathcal{K} \cup \{\alpha\})\} \subseteq \{\hat{\nu} \mid \hat{\nu} \models^3 (\mathcal{K} \cup \{\beta\})\}$. Therefore $\min_{\hat{\nu} \models^3 (\mathcal{K} \cup \{\alpha\})} |\mathsf{AffectedStates}(\hat{\nu})| \geq \min_{\hat{\nu} \models^3 (\mathcal{K} \cup \{\beta\})} |\mathsf{AffectedStates}(\hat{\nu})|$. For TS let $\mathcal{K} = \{\mathbf{X}\varphi, \mathbf{X}\neg\varphi\}$ and $\mathcal{K}' = \{\mathbf{G}\varphi, \mathbf{G}\neg\varphi\}$. Then there is an interpretation satisfying $\mathcal{K}$ that only assigns B to state t_1. But for $\mathcal{K}'$ every interpretation must assign B to m states. Therefore $\min_{\hat{\nu} \models^3 \mathcal{K}} |\mathsf{AffectedStates}(\hat{\nu})| < \min_{\hat{\nu} \models^3 \mathcal{K}'} |\mathsf{AffectedStates}(\hat{\nu})|$.

The proofs for $\mathcal{I}_c^{LTL}$ are analogous to $\mathcal{I}_d^{LTL}$ except that $\mathcal{I}_c^{LTL}$ counts the number of inconsistencies for each state also. VO and MO are immediate as before. For IN the counterexample for propositional logic can be used. DO follows as for $\mathcal{I}_d^{LTL}$. The only difference in the proof for TS is that now it is possible to give a formula φ for which $\mathcal{I}_c^{LTL}(\mathcal{K}) > 1$ but then $\mathcal{I}_c^{LTL}(\mathcal{K}') = m \times \mathcal{I}_c^{LTL}(\mathcal{K}) > \mathcal{I}_c^{LTL}(\mathcal{K})$. □

Proposition 6 (Coherence)**.** *For any $\mathcal{K}$: $\nexists \phi \in \mathcal{K}$ s.t. $+\delta\phi$ and $-\delta\phi$.*

Proof. First, observe that–by definition–for $-\delta\phi$ to hold, there needs to exist a disputee φ s.t. $+\delta\varphi$ holds, Then, to show coherence, proceed by cases. Either (1) ϕ is undisputed, so $-\delta\phi$ cannot hold by definition. Or (2) all disputors are successfully defended, so $+\delta\varphi$ cannot hold at the same time. □

Proposition 7 (Soundness)**.** *If $+\delta\phi$ then $\mathcal{K} \models \phi$.*

Proof. This follows directly from Definition 8. □

Proposition 8. *Let $\mathcal{I}$ be an inconsistency measure that satisfies CO,TS, then $S(\phi, \mathcal{I})$ satisfies $\mathsf{CO}_\mathcal{C}$, $\mathsf{BL}_\mathcal{C}$, $\mathsf{TS}_\mathcal{C}$.*

Proof. $\mathsf{CO}_\mathcal{C}$ has already been shown in [10], cf. what they call *consistency*. Also regarding [10], $\mathsf{BL}_\mathcal{C}$ follows from what they call *consistency* and *distribution*. For $\mathsf{TS}_\mathcal{C}$, observe that ϕ affects more states than φ per assumption. So the marginal contribution of ϕ will be strictly larger than for φ. □

Theorem 1. $\textsc{Upper}_{\mathcal{I}_d^{LTL}}$ *and* $\textsc{Upper}_{\mathcal{I}_c^{LTL}}$ *are* NP*-complete.*

Proof. First, observe that given a 3-valued interpretation $\hat{\nu}$, evaluating $\hat{\nu}(t_i, \phi)$ for any t_i and ϕ, as well as determining $\mathsf{AffectedStates}(\hat{\nu})$ and $\mathsf{ConflictBase}(\hat{\nu})$, can be done in polynomial time.

For NP-membership consider the following non-deterministic algorithm. Given $\mathcal{K}$ and $x \in \mathbb{R}^{\infty}_{\geq 0}$, we non-deterministically guess an interpretation $\hat{\nu}$ and verify $\hat{\nu} \models^3 \mathcal{K}$ and $|\mathsf{AffectedStates}(\hat{\nu})| \leq x$ (or $|\mathsf{ConflictBase}(\hat{\nu})| \leq x$).

For NP-hardness, observe that for given $\mathcal{K}$ and $x = 0$, both problems $\textsc{Upper}_{\mathcal{I}_d^{LTL}}$ and $\textsc{Upper}_{\mathcal{I}_c^{LTL}}$ are equivalent to the problem of deciding whether there is a two-valued interpretation that satisfies $\mathcal{K}$. Due to Proposition 2 this problem is equivalent to the classical satisfiability problem in LTL_{ff}. Therefore, we can reduce the classical satisfiability problem of propositional logic to the problem $\textsc{Upper}_{\mathcal{I}_d^{LTL}}$ (or $\textsc{Upper}_{\mathcal{I}_c^{LTL}}$) with $x = 0$, which shows the NP-hardness of the latter. $\square$

Corollary 1. *$\textsc{Lower}_{\mathcal{I}_d^{LTL}}$ and $\textsc{Lower}_{\mathcal{I}_c^{LTL}}$ are coNP-complete. $\textsc{Exact}_{\mathcal{I}_d^{LTL}}$ and $\textsc{Exact}_{\mathcal{I}_c^{LTL}}$ are in DP. $\textsc{Value}_{\mathcal{I}_d^{LTL}}$ and $\textsc{Value}_{\mathcal{I}_c^{LTL}}$ are in $\mathsf{FP}^{\mathsf{NP}[\log n]}$.*

Proof. The coNP-completeness of $\textsc{Lower}_{\mathcal{I}_d^{LTL}}$ and $\textsc{Lower}_{\mathcal{I}_c^{LTL}}$ and the membership of $\textsc{Exact}_{\mathcal{I}_d^{LTL}}$ and $\textsc{Exact}_{\mathcal{I}_c^{LTL}}$ in DP follow from Lemma 6 in [11], Proposition 1, and the fact that the measures $\mathcal{I}_c^{LTL}$ and $\mathcal{I}_d^{LTL}$ are well-serializable, cf. Definition 22 in [11], due to their range being equal to $\{0, \ldots, m, \infty\}$ (for $\mathcal{I}_d^{LTL}$) and $\{0, \ldots, m|\mathsf{At}|, \infty\}$ (for $\mathcal{I}_c^{LTL}$). The membership of $\textsc{Value}_{\mathcal{I}_d^{LTL}}$ and $\textsc{Value}_{\mathcal{I}_c^{LTL}}$ in $\mathsf{FP}^{\mathsf{NP}[\log n]}$ follow likewise from Lemma 4 in [11] and Proposition 1. $\square$

Theorem 2. *Let $\mathcal{K}$ be a knowledge base. Algorithm 1 is sound for the problem of deciding whether $\mathcal{I}_d^{LTL}(\mathcal{K})$ will be m, for any $m > 0$.*

Proof. Assuming a sequence of states $t_0, ..., t_m$, via Proposition 3, it directly follows that if an inconsistency affects $n < m$ states, the number of affected states can only grow by a constant, but can never affect all states (Via the proposition, an interpretation already exists and via $\mathcal{I}_d^{LTL}$ we have selected one with a minimal number of assignments to B). If we consider larger m, note that similar as above, as we have already selected a minimal interpretation, $\mathcal{I}_d^{LTL}$ can only increase. If the condition of line 4 does not hold, we know exactly $m+1$ states were affected before. So the only, minimal interpretation that existed was one that assigned B to all atoms in all states. Again via Proposition 3, we know that if we add a state to the sequence of states, and this state is also affected, this will hold for any length m inductively. $\square$

Theorem 3. *Let $\mathcal{K}$ be a knowledge base. Algorithm 1 is complete for the problem of deciding whether $\mathcal{I}_d^{LTL}(\mathcal{K})$ will be m, for any $m > 0$.*

Proof. Assume a sequence of states $t_0, ..., t_m$. First, note that our algorithm returns true or false on a decision, so there is no case where no solution should be reported. Then, consider the return variable. In line 4, we correctly return false if $n < m$ states are affected. So the variable is returned in this case. Then, if line 8 holds, we return true, and false otherwise. The condition in line 8 is decidable, and it is immediate to see that all values are computable. So Algorithm 1 is complete by invariance. □

Appendix B: Preliminaries on Answer Set Programming

For computing the inconsistency measures presented in this work, we develop new algorithmic approaches based on *Answer Set Programming* (ASP) [58, 59, 60], which has already been successfully applied for the computation of inconsistency values in related works [46, 35, 47, 48]. In the following, we provide a brief overview of the syntax and semantics of ASP.

Due to the declarative nature of ASP, the goal is to represent a problem in a logical format (namely, an *extended logic program*) such that the models of this representation (the *answer sets*) describe solutions of the original problem.

An extended logic program consists of *rules* which are of the form

$$r = a_0 \texttt{ :- } a_1, \ldots, a_n, \texttt{ not } a_{n+1}, \ldots, \texttt{ not } a_m. \quad (1)$$

with a_i, $i \in \{0, \ldots, n, n+1, \ldots, m\}$ being atoms. An atom is a predicate $p(v_1, \ldots, v_k)$ with $k \geq 0$, where each $v_1, \ldots, v_k$ is either a constant or a variable. If an atom does not contain any variables, it is referred to as *ground* (this concept can be applied to rules analogously). Constants are represented by strings starting with a lowercase letter, variables by strings starting with an uppercase letter[13]. We write the arity k of a predicate p as p/k. Further, in an ASP rule (Eq. 1), "`:-`" can be interpreted as "if", a "," can be read as "and", and the end of a rule is marked by a ".". Moreover, "`not`" denotes default negation in the sense of Reiter's default logic [61].

[13]Note that we also make use of *anonymous variables*. Those variables do not recur within the rule at hand, and are indicated by "_".

An ASP rule is comprised of a *head* and a *body*, which are divided by "`:-`". Hence, wrt. Eq. 1, the head consists of $\{a_0\}$, and the body of $\{a_1, \ldots, a_n, \texttt{ not } a_{n+1}, \ldots, \texttt{ not } a_m\}$. A rule with an empty body is called a *fact*, and a rule with an empty head is called a *constraint*.

Let L be a set of ground atoms. L is a *model* of a program P (i. e., a set of rules) if $a_0 \in L$ whenever $\{a_1, \ldots, a_n\} \subseteq L$ and $\{a_{n+1}, \ldots, a_m\} \cap L = \emptyset$ for each rule r included in P. The *reduct* of a program P wrt. L is defined as

$$\begin{aligned} P^L = \{a_0 \texttt{ :- } a_1, \ldots, a_n \mid \\ & a_0 \texttt{ :- } a_1, \ldots, a_n, \texttt{ not } a_{n+1}, \ldots, \texttt{ not } a_m \in P, \\ & \{a_{n+1}, \ldots, a_m\} \cap L = \emptyset\}. \end{aligned}$$

Note that the last part of this definition ensures that P^L is a *positive* program, meaning that no instance of `not` occurs in it. A positive program always possesses a uniquely defined subset-minimal model. If L is the subset-minimal model of P^L, then L is called an *answer set* of P.

In addition to the "basic" rules described above, we make use of some further language concepts that modern ASP dialects allow for. To begin with, we use *cardinality constraints*, which are of the form

$$l\{a_1; \ldots; a_n\}u$$

where l constitutes a lower bound, and u an upper bound. Thus, the above cardinality constraint can be read as "at least l, and at most u of the atoms in $\{a_1; \ldots; a_n\}$ must be included in the answer set". Note that either one of the bounds may be omitted. Moreover, ";" can be read as "or".

Furthermore, we make use of *aggregates*, which are functions that apply to sets and allow for expressing concepts such as counting, summing, or determining minima or maxima in a concise manner. To be precise, we only use a specific form of aggregates:

$$\texttt{\#agg}\{v_1, \ldots, v_n : a_1, \ldots, a_m\} = v_{n+1}$$

Here, $\texttt{\#agg} \in \{\texttt{\#count}, \texttt{\#sum}\}$ is the aggregate function name, $\{v_1, \ldots, v_n\}$ is a set of variables, $\{a_1, \ldots, a_m\}$ is a set of atoms, and v_{n+1} is a variable representing an integer value. Intuitively speaking, the set of ground instances of $\{v_1, \ldots, v_n\}$ fulfilling the conditions modeled by $\{a_1, \ldots, a_m\}$ must be equal to v_{n+1} if counted or, respectively, summed up.

Example 27. *Assume we are aiming to determine the total number of formulas in a knowledge base (expressed by* `numFormulas(K,X)`*, where* `K` *represents the knowledge base, and* `X` *the number of formulas). Let each formula be represented by* `isFormula/1` *and each knowledge base by* `isKB/1`*. We can use a* `#count` *aggregate to count all (ground) instances of* `isFormula/1` *and to state that the resulting value must be equal to* `X`*:*

```
numFormulas(X,K) :- X = #count{F: isFormula(F,K)}, isKB(K).
```

Let us now assume that we are dealing with multiple knowledge bases, and that our new goal is to determine the total number of formulas across all knowledge bases (expressed by `totalNumFormulas(Y)`*). Hence, we aim to count the number of formulas in each knowledge base and then sum up the resulting values. We can achieve this by using a* `#sum` *aggregate as follows:*

```
totalNumFormulas(Y) :- Y = #sum{X,K: numFormulas(X,K), isKB(K)}.
```

Another ASP language concept we use is the *optimization statement*. Optimization statements can express cost functions involving minimization and/or maximization. Again, we only require a specific type of optimization statement, which we define as

$$\texttt{\#minimize}\{v : a_1, \dots, a_n\}$$

with v being a variable, and $\{a_1, \dots, a_n\}$ being a set of atoms. Such a minimization statement instructs the ASP solver to find a solution in which the number of ground instances for which $\{a_1, \dots, a_n\}$ hold is minimal. We refer to an answer set that complies with the minimization as an *optimal* answer set.

Example 28. *Consider a scenario in which we already modeled the basic concepts of (propositional) logic in ASP. Our goal is now to find a solution in which a maximal number of formulas in a given knowledge base is satisfiable (i. e., in which a minimal number of fomulas is unsatisfiable). Let* `isUnsat(F)` *represent that a formula is unsatisfiable. We can find an optimal solution by using the following minimization statement:*

```
#minimize{F: isUnsat(F)}.
```