Fundamenta Informaticae 157 (2018) 1–29
DOI 10.3233/FI-2018-1600
IOS Press

Evaluating Compliance: From LTL to Abductive Logic Programming

Federico Chesani*
University of Bologna
V.le Risorgimento 2, 40136 Bologna, Italy
federico.chesani@unibo.it

Marco Gavanelli
University of Ferrara
Via Saragat 1, 44122 Ferrara, Italy
marco.gavanelli@unife.it

Evelina Lamma
University of Ferrara
Via Saragat 1, 44122 Ferrara, Italy
evelina.lamma@unife.it

Paola Mello
University of Bologna
V.le Risorgimento 2, 40136 Bologna, Italy
paola.mello@unibo.it

Marco Montali
Free Universty of Bozen-Bolzano
Piazza Domenicani 3, I-39100 Bolzano, Italy
montali@inf.unibz.it

Abstract. The *compliance verification* task amounts to establishing if the execution of a system, given in terms of observed happened events, does respect a given property. In the past both the frameworks of Temporal Logics and Logic Programming have been extensively exploited to assess compliance in different domains, such as normative multi-agent systems, business process management and service oriented computing.

In this work we review the LTL and SCIFF frameworks in the light of compliance evaluation, and formally investigate the relationship between the two approaches. We define a notion of compliance within each approach, and then we show that an arbitrary LTL formula can be expressed in SCIFF, by providing a translation procedure from LTL to SCIFF which preserves compliance.

Keywords: Linear Temporal Logic, Abductive Logic Programming, Compliance Verification.

*Address for correspondence: University of Bologna, V.le Risorgimento 2, 40136 Bologna, Italy

1. Introduction

Temporal Logic was originally developed to represent tense in natural language [1], but in Computer Science it was used for formal specification and verification of systems, initially in the realm of concurrent and distributed computing. Traditionally, Linear Temporal Logic (LTL) [2] specifications are employed for expressing the properties that a reactive system should exhibit (or avoid), and are concretely exploited by model checking tools for formal verification (e.g., [3, 4]).

In some recent research works, however, LTL formulae have been also used to declaratively describe the system under study itself. For instance, LTL has been extensively applied to this purpose in different domains, such as that of normative multi-agent systems, Business Process Management (BPM) and Service Oriented Computing (SOC). E.g., in the BPM and SOC communities, LTL formulae are used to express business rules and policies that must be respected by the interacting stakeholders and services. Even more, the DECLARE system [5] and the ConDec language [6, 7, 8] advocate the use of declarative, constraint-based specifications to model business processes with a flexible flavour, and adopt LTL for this purpose.

In these domains, a relevant task is to assess *compliance*, usually defined as checking if an implementation faithfully meets the requirements of a standard or specification. The LTL models correspond to linear Kripke structures representing the execution traces (i.e., sequences of events) occurred during a specific instantiation of the system, while entailment becomes a compliance evaluator with respect to a *regulatory specification* expressed as an LTL formula. The regulatory specification (or regulatory model) captures the desired property or behaviour to check. The algorithms for compliance therefore verify if an execution trace (or a set of execution traces) follows the prescribed behaviours or rules. Two examples of such algorithms are the one used for static compliance verification of BPMN business processes [9][1], and the one exploited in [10] for auditing event logs.

Recently, Logic Programming (LP) based approaches have been extensively applied for specification and verification of normative systems [11, 12], web services [13, 14] and business processes as well [15, 8]. The LP framework nicely meets the advantages of a declarative, first-order specification, grounded on a model-based semantics, and equipped with an operational proof procedure, computing the model itself as in Answer Set Programming (ASP), or through resolution-based methods. Therefore, compliance has been extensively investigated also in the field of logic programming, taking advantage of the language capabilities (such as variables, Constraint Logic Programming (CLP) constraints, and so on) and of the associated proof techniques.

In [16] we have defined the abductive proof procedure named SCIFF, originally developed for specification and verification of open societies of "computees" (a sort of agents), and later applied to normative systems [17, 18], web service interaction [19, 20] and BPM [21, 8]. With respect to Logic Programming, in Abductive Logic Programming (ALP) hypotheses can be made in order to prove a given goal. For example, ALP frameworks have been exploited for diagnosis purposes, by generating hypotheses that would explain observed facts. In SCIFF, ALP is used to make hypotheses about future courses of interactions, in terms of what is expected to happen, and what is expected not to happen. SCIFF specifications are given in terms of integrity constraints (a kind of IFF forward rules [22]) linking occurring (observed) events to expectations about the future course of events. SCIFF

[1] In this case, the BPMN business process intensionally represents all its possibly supported traces.

declarative semantics has been given in terms of compliance of a given trace with respect to a SCIFF specification, and the overall framework has been proved sound and complete (under proper acyclicity and boundedness assumptions for abductive logic programs, guaranteeing termination of the proof procedure).

Both the approaches based on LTL and on Logic Programming have been applied to the *compliance verification* task, each solution showing specific strengths and weaknesses. E.g., LTL-based approaches naturally deal with infinite-length system runs; the same property specification can be used to verify if the property holds with respect to an execution trace and to verify a priori if a system guarantees that property. On the contrary, for example, LP-based approaches provide a richer expressiveness, by means of variables ranging over infinite domains, and they offer the possibility of exploiting Constraint Programming techniques. Also, depending on the application domain (Multi-Agent Systems, Business Process, normative systems, Web Service Interactions), some authors prefer the term *conformance* (like e.g. in [23]); other authors in [24] distinguish between conformance and compliance depending on whether the verification task is applied to the observed execution traces (compliance) or a-priori to the system specification (conformance); finally, others authors assign a normative flavour to the terms compliance and conformance [25, 26]. Even if in this work we completely ignore normative concepts (such as, for example, obligations, duties, powers), we adopt the term *compliance*, to highlight the fact that the verification task we refer to is applied to the observed traces, and no information about the observed system is available.

In this paper we investigate the relation between the LTL-based approaches and LP-based approaches, from the specific viewpoint of the compliance. In particular, we focus on the relation between LTL and the abductive framework SCIFF, and we define a correspondence between the two approaches, in terms of the preservation of the models. The objective is not to provide a two-way bridge between them, since it would not be possible due to important differences in the expressive power. Rather, we aim to show that if we focus on the compliance, an LTL model can be (formally and correctly) translated into a SCIFF one, thus opening up the possibility of exploiting a unified framework.

Starting from the seminal work by Fisher et al. [27] about Separated Normal Forms (SNF) for LTL formulae, we define proper mapping functions and show how any LTL formula can be expressed within the SCIFF formalism. Then, we formally define the notion of compliance in both approaches, and we discuss the equivalence of the two definitions from the compliance perspective. Finally, we prove how such equivalence is indeed maintained when formula/property specifications are translated from the LTL approach to the SCIFF-based one.

The contribution of this work is mainly theoretical: we prove that for *any* LTL formula φ, models (w.r.t. the notion of compliance) are preserved from LTL to LP (and abduction, specifically). From the practical viewpoint however, two issues arise: firstly, compliance is decidable in propositional LTL, while LP (and abduction) is semi-decidable. Secondly, the SNF transformation introduces formulas with an inductive flavour: therefore, boundedness conditions for a SCIFF program are not met, and termination is not guaranteed. Roughly speaking, this is the price that our translation pays for proving model preservation for *any* LTL formula. However, we might point out that real applications often can be represented with a small, yet significant fragment of LTL: in [20] we investigated how practical business constraints (whose semantics was given in terms of an LTL fragment) can be automatically

translated into SCIFF and, more important, can be reasoned upon in an abductive framework. In [28], performance tests showed how an LP-based approach was comparable, under the performances perspective, with LTL-based model checking methods. In this light, this current work can be seen as a complement to the cited previous works: if in the latter works the focus was on practical applicability and performances, in this work we investigate more semantic aspects, and specifically if and how models can be preserved between the two logics.

This paper is organized as follow: in Sections 2 and 3 we introduce LTL-based and SCIFF-based approaches, respectively. In Section 4 we provide a formal definition of compliance equivalence between the approaches, and then in Section 5 we introduce an automatic method to translate LTL formulae into SCIFF formulae, preserving the notion of compliance equivalence. Finally, in Sections 6 and 7 we discuss related works and conclude.

2. Linear Temporal Logic

In this section, we provide a brief introduction to (propositional) Linear-time Temporal Logic (LTL), in particular with respect to the notion of *compliance*, as introduced in Section 1; the interested reader can refer to [2] for a more general introduction to LTL.

LTL formulae are built up from *atomic propositions*, whose truth values change over time. In general, LTL denotes a temporal logic that is:

linear: each situation has a single next future moment, and temporal modalities therefore predicate on the truth of propositions along a single timeline;

qualitative: Temporal operators are used to express qualitative time relations between propositions. Metric distances are not part of the logic;

point-based: operators and propositions are evaluated over *points* in time;

discrete: the present moment corresponds to the current state of the system and the next one to its immediate successor (where the first future event happens);

future-tense: temporal operators refer to the occurrence of events in the future.

2.1. LTL models and execution traces

In accordance with the outlined properties, an LTL *time structure* $\mathcal{F}$, also called *frame*, models a single, linear timeline; formally, $\mathcal{F}$ is a totally ordered set $(\mathcal{K}, \prec)$ [2]. Usually, $\mathcal{K}$ is the infinite set of natural numbers $\mathbb{N}$.

Definition 2.1. (LTL model)
Let $\mathcal{P}$ be the set of all atomic propositions in the system. An LTL model $\mathcal{M}$ for $\mathcal{P}$ is a triple $(\mathcal{K}, \prec, \mathtt{v})$ where $\mathtt{v} : \mathcal{P} \rightarrow 2^{\mathcal{K}}$ is a *function* which maps each proposition in $\mathcal{P}$ to the set of time instants at which the proposition holds.

We are interested in systems characterized by dynamics consisting of a stream of events. In this respect, events occurring in an instance of the system can be represented by atomic propositions. Therefore, we will restrict our attention to the set $\mathcal{E}$ of atomic propositions e representing events. The dynamic evolution of a system can be represented by which propositions are true at which time instant: i.e., for each time instant i there is a corresponding *state* defined as the set of all propositions that are true in the time instant i. Under this interpretation, LTL models correspond to execution traces.

Definition 2.2. (LTL execution trace)
Given a set $\mathcal{E}$ of atomic propositions (representing possible events), an *LTL execution trace* $\mathcal{T}_{\text{LTL}}$ is an LTL model having $(\mathbb{N}, <)$ as time structure and $\mathcal{E}$ as the set of atomic propositions. In particular, $\mathcal{T}_{\text{LTL}} = (\mathbb{N}, <, v_{occ})$, where $v_{occ} : \mathcal{E} \to 2^{\mathbb{N}}$ is a valuation function mapping each event $e \in \mathcal{E}$ to the set of all time instants $i \in \mathbb{N}$ at which e occurs.

For convenience, we will sometimes use the following abbreviations: $\mathcal{T}_{\text{LTL}}(i)$ will denote the i-th state of $\mathcal{T}_{\text{LTL}}$, and $e \in \mathcal{T}_{\text{LTL}}(i)$ will be adopted as a synonym for $i \in v_{occ}(e)$. Notice that propositions in the i-th state can be occurred events, or can be logical consequences of LTL formulas. Moreover, we currently have that $\mathcal{P} \equiv \mathcal{E}$: however in Section 5.1 we introduce the SNF transformation that will add further atomic propositions not related to events.

2.2. Syntax of LTL

LTL formulae are defined by using (i) *atomic propositions*, i.e., events, together with the two special constants $true$ and $false$; (ii) *classical propositional connectives*, i.e., $\neg$, $\wedge$, $\vee$ and $\Rightarrow$; (iii) *temporal operators*, i.e., $\bigcirc$ (next time), $\mathcal{U}$ (until), $\Diamond$ (eventually), $\Box$ (globally) and $\mathcal{W}$ (weak until).

Starting from these constitutive elements, an LTL formula is recursively defined as: each event $e \in \mathcal{E}$ is a formula; if φ and ψ are formulae, then $\neg\varphi$, $\varphi \wedge \psi$, $\bigcirc\psi$, and $\varphi\mathcal{U}\psi$ are formulae.

Other LTL formulae can be defined as abbreviations:

- $\varphi \vee \psi \triangleq \neg(\neg\varphi \wedge \neg\psi)$ and $\varphi \Rightarrow \psi \triangleq \neg\varphi \vee \psi$;
- $true \triangleq \neg\varphi \vee \varphi$ and $false \triangleq \neg true$;
- $\Diamond\varphi \triangleq true\mathcal{U}\varphi$;
- $\Box\varphi \triangleq \neg\Diamond\neg\varphi$;
- $\psi\mathcal{W}\varphi \triangleq \psi\mathcal{U}\varphi \vee \Box\psi$.

Temporal operators have maximum priority, then $\neg$, $\wedge$, $\vee$ and finally $\Rightarrow$.

2.3. Semantics of LTL and compliance

The semantics of LTL is given with respect to an LTL execution trace, and with respect to a specific state. We will use $\models_{\text{LTL}}$ to denote the logical *entailment* in the LTL setting. $\mathcal{M}, i \models_{\text{LTL}} \varphi$ means that φ is true at time i in model $\mathcal{M}$. $\models_{\text{LTL}}$is defined by induction on the structure of the formulae[2]:

[2]For the sake of readability, we explicitly show the semantics of $\Diamond$, $\Box$ and $\mathcal{W}$, even if their semantics can be obtained from the semantics of $\mathcal{U}$ and $\Box$.

$(\mathcal{T}_{\text{LTL}} \models_{\text{LTL}} \varphi)$ iff $(\mathcal{T}_{\text{LTL}}, 0 \models_{\text{LTL}} \varphi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} e)$ iff $e \in \mathcal{T}_{\text{LTL}}(i)$ (i.e., $i \in v_{occ}(e)$);

$(\mathcal{T}_{\text{LTL}}, i \not\models_{\text{LTL}} e)$ iff $e \notin \mathcal{T}_{\text{LTL}}(i)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \neg\varphi)$ iff $(\mathcal{T}_{\text{LTL}}, i \not\models_{\text{LTL}} \varphi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \varphi \wedge \psi)$ iff $(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \varphi)$ and $(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \psi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \varphi \vee \psi)$ iff $(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \varphi)$ or $(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \psi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \varphi \Rightarrow \psi)$ iff $(\mathcal{T}_{\text{LTL}}, i \not\models_{\text{LTL}} \varphi)$ or $(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \psi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \bigcirc\varphi)$ iff $(\mathcal{T}_{\text{LTL}}, i+1 \models_{\text{LTL}} \varphi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \psi\mathcal{U}\varphi)$ iff $\exists k \geq i$ s.t. $(\mathcal{T}_{\text{LTL}}, k \models_{\text{LTL}} \varphi)$ and $\forall j. i \leq j < k$ $(\mathcal{T}_{\text{LTL}}, j \models_{\text{LTL}} \psi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \Diamond\varphi)$ iff $\exists j \geq i$ s.t. $(\mathcal{T}_{\text{LTL}}, j \models_{\text{LTL}} \varphi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \Box\varphi)$ iff $\forall j \geq i$ $(\mathcal{T}_{\text{LTL}}, j \models_{\text{LTL}} \varphi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \psi\mathcal{W}\varphi)$ iff either $(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \psi\mathcal{U}\varphi)$ or $(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \Box\psi)$.

Note that the $\mathcal{W}$ operator relaxes $\mathcal{U}$ by removing the requirement that φ must eventually hold in a future moment, and by stating that in this case ψ must hold in all the future states.

When LTL is employed to formalize compliance rules, the declarative semantics selects those events that must be contained (or avoided) in certain states so as to fulfill them, separating compliant traces from non-compliant ones. In this respect, $\models_{\text{LTL}}$ plays the role of a *compliance evaluator*.

Definition 2.3. (LTL Compliance)
An LTL trace $\mathcal{T}_{\text{LTL}}$ is *compliant* with an LTL formula φ if and only if $\mathcal{T}_{\text{LTL}}$ entails φ:

$$\text{CMP}_{\text{LTL}}(\mathcal{T}_{\text{LTL}}, \varphi) \triangleq \mathcal{T}_{\text{LTL}} \models_{\text{LTL}} \varphi.$$

When LTL formulae are used to express business constraints/rules of a regulatory model, as for example in the ConDec language [6], then the LTL formula used for compliance is the conjunction of all formulae contained in the regulatory model. From an operational viewpoint, the compliance of a formula φ with respect to a trace $\mathcal{T}_{\text{LTL}}$ is verified by means of model checking algorithms.

3. The SCIFF framework

In this section, we review the language and semantics of the SCIFF framework, showing how compliance is grounded on expectations, traces and fulfillment. For a comprehensive description of the SCIFF framework see [16].

3.1. SCIFF specifications and traces

A SCIFF specification $\mathcal{S}$ is an Abductive Logic Program $\langle \mathcal{KB}, \mathcal{A}, \mathcal{IC} \rangle$ [29] where: (i) $\mathcal{KB}$ is a (static) knowledge base, i.e., a Logic Program (we rely on standard LP terminology [30]); (ii) $\mathcal{A}$ is a special set of predicates, called *abducibles*; two special abducibles, namely $\mathbf{E}/2$ and $\mathbf{EN}/2$, are used to represent the *expectations*; (iii) $\mathcal{IC}$ is a set of SCIFF integrity constraints, relating happened events with expectations.

$\mathcal{KB}$ constitutes a deductive knowledge base representing the static aspects of the domain under study, while $\mathcal{IC}$ represents an abductive knowledge, composed of a set of declarative rules (integrity constraints) used to constrain the behaviour of the interacting entities when executing an instance of the system[3]. Usually in Abductive Logic Programming the $\mathcal{KB}$ is exploited by *backward* reasoning methods, while $\mathcal{IC}$ is exploited in a *forward* manner.

Roughly speaking, given a goal $\mathcal{G}$[4], abductive reasoning looks for a set of literals Δ built from predicates in $\mathcal{A}$ such that the goal $\mathcal{G}$ and the set $\mathcal{IC}$ of integrity constraints are entailed by the program $\mathcal{KB} \cup \Delta$. The set Δ is referred to as an *abductive explanation* (see Definition 3.3). From another viewpoint, the abductive reasoning process consists of making hypotheses so to entail the goal; the allowed hypotheses are built from predicates which must belong to $\mathcal{A}$, and should respect some constraints ($\mathcal{IC}$).

Three special predicates are used to model happened events and positive/negative expectations, which in turn characterize occurred and expected courses of interaction. Happened events are denoted by using the (non abducible) predicate $H(E, T)$, where E is a placeholder for a term representing the occurred event, while the placeholder T explicitly represents the time at which the event occurred[5]. Different time domains can be used to characterize T, but in the remainder of this paper we will rely on the natural numbers. Hence, within the SCIFF framework, the execution trace of a system corresponds to a set of ground $H(E, T)$ atoms.

Definition 3.1. (SCIFF Execution Trace)
A SCIFF *execution trace* $\mathcal{T}_{\text{SCIFF}}$ (or simply a SCIFF trace) is a set of ground $H(E, T)$ atoms.

A specific execution of the system under study is called an *instance*, and it is formally identified by the SCIFF specification modelling the system and by the execution trace produced during the instance execution.

Definition 3.2. (SCIFF Instance)
Given a SCIFF specification $\mathcal{S} = \langle \mathcal{KB}, \mathcal{A}, \mathcal{IC} \rangle$ and a trace $\mathcal{T}_{\text{SCIFF}}$, $\langle \mathcal{S}, \mathcal{T}_{\text{SCIFF}} \rangle$ is an *instance* of $\mathcal{S}$.

Positive and negative expectations model instead expected and forbidden events. They are represented by $\mathbf{E}(E, T)$ and $\mathbf{EN}(E, T)$, where E is a term describing the event, and T is a term. The intended meaning is that event E is expected to occur/not occur at time T.

[3] Since the knowledge base is not relevant for the translation from LTL to SCIFF, we will mainly focus on integrity constraints.

[4] Notice that, w.r.t. [30], here we call *goal* the logical formula to be proved.

[5] The SCIFF framework aims to monitor running systems, whose internals are not observable. To this end, SCIFF observes the system by means of *events* that *happen* at certain *time instants*.

SCIFF Integrity Constraints (ICs) are mainly used to relate happened events with expectations. They are $body \rightarrow head$ rules, where $body$ contains a conjunction of happened events and abducibles, while $head$ contains a disjunction of conjunctions of expectations and abducibles. When the body is matched with events and abducibles, the Integrity Constraint is triggered, and expectations occuring in the head are assumed (abduced). All the constitutive elements of an integrity constraint may contain variables. Roughly speaking, variables contained inside a happened event are universally quantified with scope the entire rule, modelling that all ground event occurrences match with the happened event occurring in the rule. Variables appearing in a positive expectation $\mathbf{E}(E, T)$ are existentially quantified, since a positive expectation requires that some event matching with E will occur at a time matching with T. Conversely, variables appearing in negative expectations are universally quantified, forbidding the execution of all matching event occurrences[6]. Both variables contained into happened events and expectations can occur inside predicates defined in the static $\mathcal{KB}$, or subject to Constraint Logic Programming (CLP) constraints [31]. Although the SCIFF framework does not have such restriction, in the following we assume that CLP constraints will be only linear inequalities over natural numbers. When applied to time variables, CLP constraints are used to express both qualitative and quantitative time constraints. For example, the SCIFF integrity constraint:

$$H(a, T) \rightarrow \mathbf{E}(b, T_2) \wedge T_2 > T \wedge T_2 < T + 10 \wedge \mathbf{EN}(c, T_3) \wedge T_3 > T \wedge T_3 < T_2.$$

states that every occurrence of event *a* at any time T requires that *b* will occur *afterwards* (at any time T_2 greater than T) and within 10 time units at most, and that no event *c* occurs in between (i.e., at any time T_3 between T and T_2).

The expressive power of the SCIFF language is not limited to represent propositional events. For example, the following integrity constraint:

$$H(a(V, D), T) \wedge V < 100 \rightarrow \mathbf{E}(b(V), T_2) \wedge T_2 > T \wedge T_2 < T + D.$$

states that any occurrence of an event described by $a(V, D)$ with value V less than 100, at any time T (V and D variables) requires that an event $b(V)$ will occur after the time instant T, and within D time units at most. Here the variables in the body of the rule are used to provide further requirements on *what* is expected to happen (event b must have the same V of event a) and *when*.

3.2. SCIFF and abduction

As pointed out by Torroni et al. [32], in open Multiagent Systems (but the same holds for any dynamic environment in which events occur over time)

> "a large share of information is unknown: either because it is not specified or it is private knowledge, or – especially in the case of events – because it has not (yet) been observed by anyone or is still to occur. To capture the unknown, the reasoning conducted to manipulate expectations is intrinsically hypothetical, and the social semantics of expectations is based on abductive reasoning."

[6]For a detailed and formal definition of variable quantification and scope within integrity constraints the reader can refer to [16].

This motivates why expectations are interpreted as hypotheses (i.e., abducibles) on the possible correct evolutions of the system. However, being able to generate hypotheses might not be enough: in specific domains like, e.g., legal reasoning, a further step of verification of the hypotheses against the observed events (available data) is mandatory.

Hence, the SCIFF declarative semantics does not only provide an abductive interpretation to expectations, as done for generic abducibles. It goes further, capturing the specific meaning carried by expectations linking them with happened events through the notions of *fulfillment* and *violation*. Indeed, alongside the generation of hypotheses about the correct courses of interaction as integrity constraints trigger, an *hypotheses-confirmation* check is performed, to evaluate if the actual execution trace of the system adheres to the generated expectations. This constitutes the basis for giving a formal account of the notion of *compliance* of an execution trace with a regulatory model.

Therefore, the declarative semantics of SCIFF is presented in the following in two parts. First, being SCIFF specifications formalized by means of abductive logic programs, the abductive explanations of such programs is presented. Then, we formally define expectations, providing the basis for compliance.

Definition 3.3. (Abductive explanation Δ)
Given a SCIFF instance $\langle \mathcal{S}, \mathcal{T}_{\text{SCIFF}} \rangle$, a set $\Delta \subseteq \mathcal{A}$ is an *abductive explanation* for $\langle \mathcal{S}, \mathcal{T}_{\text{SCIFF}} \rangle$ if and only if

$$\mathit{Comp}\left(\mathcal{KB} \cup \mathcal{T}_{\text{SCIFF}} \cup \Delta\right) \cup \text{CET} \cup T_{\mathcal{X}} \models \mathcal{IC}$$

where *Comp* is the (three-valued) completion of a theory [33], CET stands for Clark Equational Theory [34] and $T_{\mathcal{X}}$ is the CLP constraint theory [35], parametrized by the domain $\mathcal{X}$.

Notice that a three-value completion is adopted in SCIFF to support compliance checking in the setting of open, dynamically growing trace. If we consider finite traces, SCIFF semantics can be referred refer to a two-value completion as in [34]: in such a setting, negation can be intended as Negation As Failure (NAF).

We remind for completeness that CET is provided by the following axioms:

- for any pair of distinct constants c, c' : $\quad c \neq c'$
- for any pair of distinct functors f, g : $\quad f(x_1, \ldots, x_n) \neq g(y_1, \ldots, y_m)$
- $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \rightarrow x_1 = y_1 \wedge \ldots x_n = y_n$
- for any functor f and for any constant c : $\quad f(x_1, \ldots, x_n) \neq c$
- for any term structure $\tau(x)$ in which x is free: $\quad \tau(x) \neq x$
- for any formula W : $\quad x = y \rightarrow [W(x) \leftrightarrow W(y)]$

together with the usual rules of reflexivity, symmetry and transitivity for equality.

Fixing a CLP theory corresponds to instantiating the parameter $\mathcal{X}$ and the set of allowed constraints. Therefore, different structures can be chosen without affecting the notion of SCIFF's abductive explanation. We will instantiate such a parameter to $\mathbb{N}$, with linear equations and inequalities. The

theory of constraints $T_{\mathbb{N}}$ defines the symbols $+, -, *, /$(division between integers), $=, >, <, \geq, \ldots$ with the usual meanings (e.g., $1 < 2 + 2$ is evaluated to *true*).

It is worth noting that if a set of integrity constraints $\mathcal{IC}$ is entailed, then each single constraint is entailed as well, as stated by the following remark.

Remark 3.4. (Abductive explanations and sub-specifications)
If Δ is an abductive explanation for $\langle \mathcal{S}, \mathcal{T}_{\text{SCIFF}} \rangle$ where $\mathcal{S} = \langle \mathcal{KB}, \mathcal{A}, \mathcal{IC} \rangle$, then Δ is an abductive explanation also for $\langle \langle \mathcal{KB}, \mathcal{A}, \mathcal{IC}' \rangle, \mathcal{T}_{\text{SCIFF}} \rangle$, where $\mathcal{IC}' \subseteq \mathcal{IC}$.

3.3. SCIFF declarative semantics and compliance

As we have seen, expectations are special abducible predicates, which carry a prescriptive meaning: positive (negative) expectations state that the involved event is expected to (not) happen. Hence, expectations are strongly connected to the happened events characterizing a system's instance: depending on the actual event occurrences contained in a trace, they become *fulfilled* or *violated*, attesting the compliance or non-compliance of the trace. The SCIFF declarative semantics aims at capturing this intended meaning, making explicit the relationships between positive and negative expectations, and between expectations and happened events.

First of all, expectations must be **E**-*consistent*: the same event cannot be expected to happen and not to happen at the same time.

Definition 3.5. (E-consistency)
An abducible set Δ is **E**-*consistent* iff for each event e and for each time t it holds that $\{\mathbf{E}(e,t), \mathbf{EN}(e,t)\} \not\subseteq \Delta$.

The relationship between expectations and happened events is instead captured by the notion of *fulfillment*, which is guaranteed if a positive (negative) expectation does (not) have a matching happened event inside the trace.

Definition 3.6. (Fulfillment)
Given a SCIFF trace $\mathcal{T}_{\text{SCIFF}}$ and an abducible set Δ, Δ is $\mathcal{T}_{\text{SCIFF}}$*-fulfilled* iff for each event e and for each time t: $\mathbf{E}(e,t) \in \Delta \rightarrow H(e,t) \in \mathcal{T}_{\text{SCIFF}}$ and $\mathbf{EN}(e,t) \in \Delta \rightarrow H(e,t) \notin \mathcal{T}_{\text{SCIFF}}$.

Similarly, the violation is defined as follows:

Definition 3.7. (Violation)
Given a SCIFF trace $\mathcal{T}_{\text{SCIFF}}$ and an abducible set Δ, Δ is $\mathcal{T}_{\text{SCIFF}}$*-violated* iff it exists at least one event e and time t such that: $\mathbf{E}(e,t) \in \Delta \wedge H(e,t) \notin \mathcal{T}_{\text{SCIFF}}$, or $\mathbf{EN}(e,t) \in \Delta \wedge H(e,t) \in \mathcal{T}_{\text{SCIFF}}$.

Since expectations are abducibles, fulfilment can be interpreted as a particular form of *hypotheses confirmation* [36, 37]:

- Expectations are hypothesized according to the instance of the system and the SCIFF specification; this leads to the formulation of expectations as abductive explanations, according to Def. 3.3.

- According to Def. 3.6, the execution trace may fulfill or violate the generated expectations or not, confirming or disconfirming the formulated hypotheses.

Given an abductive explanation Δ, fulfillment acts as a classifier that separates the legal/correct execution traces with respect to Δ from the wrong ones. A trace is correct if it satisfies all the expectations contained in Δ, while it is wrong if an expected event is omitted, or a forbidden event is present. In this light, the declarative semantics of SCIFF formally characterizes the notion of *compliance* of an execution trace with a given regulative specification, where fulfillment isolates, in the space of all execution traces, those that are supported by the specification.

Definition 3.8. (Compliance in SCIFF)
A trace $\mathcal{T}_{\text{SCIFF}}$ is *compliant* with a SCIFF specification $\mathcal{S}$ if and only if there exists an abducible set Δ such that:

1. Δ is an abductive explanation for $\langle \mathcal{S}, \mathcal{T}_{\text{SCIFF}} \rangle$;
2. Δ is **E**-consistent;
3. Δ is $\mathcal{T}_{\text{SCIFF}}$-fulfilled.

If this is the case, we write $\text{CMP}_{\text{SCIFF}}{}^{\Delta}\left(\mathcal{T}_{\text{SCIFF}}, \mathcal{S}\right)$ or simply $\text{CMP}_{\text{SCIFF}}\left(\mathcal{T}_{\text{SCIFF}}, \mathcal{S}\right)$.

4. Relating LTL and SCIFF

As we have seen in Sect. 2.3 and 3.2, even if LTL and SCIFF rely on different logics, when capturing regulatory models they both act as compliance evaluators. Each framework has its own notion of compliance, nevertheless both approaches capture the same idea of compliance. We now provide a formal account of such a similarity, introducing a mapping between LTL and SCIFF execution traces, and a notion of "behavioural equivalence" between the two frameworks with respect to compliance.

4.1. Trace mapping

A very intuitive mapping can be introduced to map an LTL trace $\mathcal{T}_{\text{LTL}}$ onto a corresponding SCIFF trace $\mathcal{T}_{\text{SCIFF}}$ and vice versa: the states of an LTL trace are mapped onto time instants over $\mathbb{N}$ in a corresponding SCIFF trace, where the fact that e is executed in state t is represented as $H(e,t)$.

Definition 4.1. (Trace mapping)
Given an LTL trace $\mathcal{T}_{\text{LTL}} = (\mathbb{N}, <, v_{occ})$ and the set of atomic propositions $\mathcal{E}$, we map any possible pair (e,i) into a corresponding SCIFF event $H(e,i)$, where $e \in \mathcal{E}$ and $i \in \mathbb{N}$, (note that such event mapping is a bijective function).

A *trace mapping* $\mathtt{tm}$ is a transformation which maps an arbitrary LTL trace $\mathcal{T}_{\text{LTL}}$ onto a corresponding SCIFF one, by applying the event mapping to each proposition belonging to $\mathcal{T}_{\text{LTL}}$, i.e. to each $e \in \mathcal{E}$ and for each $i \in v_{occ}(e)$:

$$\mathtt{tm}(\mathcal{T}_{\text{LTL}}) = \left\{ H(e,i) | e \in \mathcal{E}, i \in v_{occ}(e) \right\}$$

Example 4.2. (Trace mapping)
Let us consider an LTL execution trace $\mathcal{T}_{\text{LTL}} = (\mathbb{N}, <, v_{occ})$, where $\mathcal{E} = \{a, b, c, d\}$ is the set of propositional events and v_{occ} is defined as follows:

$$v_{occ}(a) = \{0, 1\} \qquad v_{occ}(b) = \{2\} \qquad v_{occ}(c) = \{3\} \qquad v_{occ}(d) = \emptyset$$

Then $\texttt{tm}\left(\mathcal{T}_{\text{LTL}}\right) = \left\{\ H(a,0),\ H(a,1),\ H(b,2),\ H(c,3)\ \right\}$

Note that, being `tm` based on the bijective function mapping the events, it can be applied back and forth. The inverse translation, which starts from a SCIFF execution trace and produces a corresponding LTL trace, will be denoted by $\texttt{tm}^{-1}$.

4.2. Behavioural equivalence

Thanks to the trace mapping function `tm`, it is possible to evaluate whether the "same" execution trace complies with an LTL and a SCIFF specification: if the two regulatory models agree, then they express in some sense "equivalent" prescriptions with respect to the trace. Generalizing, if such an agreement is valid for all the possible execution traces, then the two specifications are *behaviorally equivalent*.

Definition 4.3. (Behavioural equivalence w.r.t. compliance)
A SCIFF specification $\mathcal{S}$ and an LTL formula φ are *behaviorally equivalent with respect to compliance* ($\varphi \overset{c}{\leftrightsquigarrow} \mathcal{S}$) if and only if *for each* LTL trace $\mathcal{T}_{\text{LTL}}$ it holds that:

$$\text{CMP}_{\text{LTL}}\left(\mathcal{T}_{\text{LTL}}, \varphi\right) \Longleftrightarrow \text{CMP}_{\text{SCIFF}}\left(\texttt{tm}\left(\mathcal{T}_{\text{LTL}}\right), \mathcal{S}\right).$$

We might notice that Definition 4.3 does not pose any constraint on the SCIFF specification $\mathcal{S}$: indeed, only the trace $\mathcal{T}_{\text{LTL}}$ is somehow constrained by the application of the mapping function `tm`. This captures the fact that the compliance goal is to rule in or out traces: the following Section 5 will introduce an automatic way for generating a proper SCIFF specification $\mathcal{S}$ that guarantees the behavioural equivalence.

5. On the expressiveness of SCIFF

Starting from the similarities of LTL and SCIFF when evaluating compliance, we now formally investigate the relationship between the two approaches. In particular, we show that an arbitrary LTL formula can be expressed in SCIFF by providing an automatic translation procedure from LTL to SCIFF which preserves the compliance equivalence. To this end, we exploit the Separated Normal Form (SNF) introduced by Fisher et al. [27] for LTL formulae.

5.1. A separated normal form for LTL formulae

Fisher et al. [27] introduced SNF to express an arbitrary LTL formula by adopting a conjunction of three basic forms, while preserving satisfiability.

Definition 5.1. (SNF Formula [27])
An LTL formula φ is in *SNF* iff φ is a conjunction of formulas of the following forms:

$$\mathbf{START} \implies \bigvee_c l_c \qquad \text{(an initial LTL-clause)}$$

$$\Box\Big(\bigwedge_a k_a \implies \bigcirc \bigvee_d l_d\Big) \qquad \text{(a step LTL-clause)}$$

$$\Box\Big(\bigwedge_b k_b \implies \Diamond l\Big) \qquad \text{(a sometime LTL-clause)}$$

where k_i and l_j are literals (i.e., atomic propositions or negation of atomic propositions) and **START** is a special symbol true only at the initial time (i.e., whose valuation function is the set $\{0\}$). In this case, we say that φ is an SNF formula.

Definition 5.2. (LTL to SNF translation [27])
`snf` is a function which translates an arbitrary LTL formula to a corresponding SNF formula. The transformation rules are given in [27][7].

Computational costs for the `snf` function are known: the number of introduced clauses is polynomial in the size of the original formula (Theorem 7.1.2 in [27]). The same holds for the number of introduced propositional symbols, again polynomial w.r.t. to the size of the initial formula (Theorem 7.1.4 in [27]).

During the transformation, new proposition symbols are introduced to rename complex sub-formulae and to allow the translation of the $\mathcal{U}$ operator. As a consequence, `snf` preserves satisfiability but not equivalence (due to the presence of these new propositional symbols). We differentiate, among the whole set of proposition symbols, the subset used to represent activities/events, and the subset introduced by `snf` for renaming.

Definition 5.3. (Proposition symbols, renaming and event sets)
Given an LTL formula φ, $\mathcal{P}(\varphi)$ is the set of proposition symbols contained in φ. Given an SNF formula σ s.t. $\sigma = \texttt{snf}(\varphi)$, it holds that $\mathcal{P}(\sigma) = \mathcal{E}(\sigma) \cup \mathcal{R}(\sigma)$, where:

1. *event set* $\mathcal{E}(\sigma)$ is the set of atomic propositions contained in the original LTL formula φ, which denote events ($\mathcal{E}(\sigma) = \mathcal{P}(\varphi)$)

2. *renaming set* $\mathcal{R}(\sigma)$ is the set of atomic propositions introduced by `snf` for renaming during the transformation.

Example 5.4. (SNF representation of the "precedence" formula)
Let us consider LTL "precedence" formula stating that the send_receipt activity can be executed only after having executed the pay activity:

$$\varphi = \neg send\_receipt\ \mathcal{W}\ pay$$

[7]Note that in [27] the `snf` function is called τ.

Hence, $\mathcal{P}(\varphi) = \{pay, send\_receipt\}$. The SNF translation of φ is:

$$
\begin{array}{rcl}
\sigma & = & \mathtt{snf}\,[\neg send\_receipt\ \mathcal{W}\ pay] = \\
& = & \mathbf{START} \Rightarrow \mathbf{X} \wedge \quad \mathtt{snf}\,[\mathbf{X} \Rightarrow (\neg send\_receipt\ \mathcal{W}\ pay)] = \\
& = & \mathbf{START} \Rightarrow \mathbf{X} \wedge \left\{ \begin{array}{l} \mathtt{snf}\,[\mathbf{X} \Rightarrow (\neg send\_receipt \vee pay)] \wedge \\ \mathtt{snf}\,[\mathbf{X} \Rightarrow (\mathbf{Y} \vee pay)] \wedge \\ \mathbf{Y} \Rightarrow \bigcirc\,(\neg send\_receipt \vee pay) \wedge \\ \mathbf{Y} \Rightarrow \bigcirc\,(\mathbf{Y} \vee pay) \end{array} \right. = \\
& = & \mathbf{START} \Rightarrow \mathbf{X} \wedge \left\{ \begin{array}{l} \mathbf{START} \Rightarrow (\neg \mathbf{X} \vee \neg send\_receipt \vee pay) \wedge \\ \mathbf{TRUE} \Rightarrow \bigcirc\,(\neg \mathbf{X} \vee \neg send\_receipt \vee pay) \wedge \\ \mathbf{START} \Rightarrow (\neg \mathbf{X} \vee \mathbf{Y} \vee pay) \wedge \\ \mathbf{TRUE} \Rightarrow \bigcirc\,(\neg \mathbf{X} \vee \mathbf{Y} \vee pay) \wedge \\ \mathbf{Y} \Rightarrow \bigcirc\,(\neg send\_receipt \vee pay) \wedge \\ \mathbf{Y} \Rightarrow \bigcirc\,(\mathbf{Y} \vee pay) \end{array} \right.
\end{array}
$$

Therefore, $\mathcal{R}(\sigma) = \{\mathbf{START}, \mathbf{X}, \mathbf{Y}, \mathbf{TRUE}\}$.

5.2. Translation from SNF formulae to SCIFF

We now provide a syntactic procedure which translates an arbitrary SNF formula to SCIFF, and prove that such a translation preserves compliance.

Definition 5.5. ($\mathcal{IC}$-mapping)
An $\mathcal{IC}$-mapping `icm` is a function which translates an SNF formula to a set of SCIFF integrity constraints, and is defined as follows[8]:

$$
\begin{array}{rcl}
\mathtt{icm}\left[\bigwedge_i \varphi_i\right] & \triangleq & \bigcup_i \mathtt{icm}\,[\varphi_i] \\
\mathtt{icm}\left[\mathbf{START} \implies \bigvee_c l_c\right] & \triangleq & \mathtt{icm}\,[start, 0] \rightarrow \bigvee_c \mathtt{icm}\,[l_c, 0]\,. \\
\mathtt{icm}\left[\Box\left(\bigwedge_a k_a \implies \bigcirc \bigvee_d l_d\right)\right] & \triangleq & \bigwedge_a \mathtt{icm}\,[k_a, T] \rightarrow \bigvee_d \left(\mathtt{icm}\,[l_d, T_2] \wedge T_2 = T + 1\right). \\
\mathtt{icm}\left[\Box\left(\bigwedge_a k_a \implies \Diamond l\right)\right] & \triangleq & \bigwedge_a \mathtt{icm}\,[k_a, T] \rightarrow \mathtt{icm}\,[l, T_2] \wedge T_2 \geq T. \\
\mathtt{icm}\,[start, 0] & \triangleq & \mathbf{OCC}(start, 0) \\
\mathtt{icm}\,[true, T] & \triangleq & \mathbf{TRUE}(T) \\
\mathtt{icm}\,[a, T] & \triangleq & \mathbf{OCC}(a, T) \\
\mathtt{icm}\,[\neg a, T] & \triangleq & \mathbf{NOT\_OCC}(a, T)
\end{array}
$$

[8] Abducible predicates will be represented as **bold** terms.

Where a stands for a generic propositional symbol. The $\mathcal{IC}$-mapping maps the presence of a certain proposition in a given state onto an abducible **OCC**/2, which states that the proposition *occurs* in that state. Conversely, the absence of the proposition is mapped onto an abducible **NOT_OCC**/2, which expresses that the proposition does *not* occur in that state. Quantification of variables is the same given for SCIFF integrity constraints, as mentioned in Section 3.1.

Definition 5.6. ($\mathcal{S}$-mapping sm)
Given an SNF formula φ and a set $\mathcal{V} \subseteq \mathcal{P}(\varphi)$ of proposition symbols, the *$\mathcal{S}$-mapping* sm translates φ to a SCIFF specification depending on $\mathcal{V}$.
sm is defined as:

$$\mathtt{sm}\,[\varphi, \mathcal{V}] \triangleq \langle \emptyset, \{\mathbf{E}/2, \mathbf{EN}/2, \mathbf{TRUE}/1, \mathbf{OCC}/2, \mathbf{NOT\_OCC}/2\}, \mathcal{IC}) \rangle$$

where

$$\begin{aligned}
\mathcal{IC} = \mathtt{icm}(\varphi) \cup \{ & \\
& true \rightarrow \mathbf{OCC}(start, 0). && (S)\\
& true \rightarrow \mathbf{TRUE}(0). && (T_1)\\
& \mathbf{TRUE}(T) \rightarrow \mathbf{TRUE}(T_2) \wedge T_2 = T + 1. && (T_2)\\
& \forall p \in \mathcal{P}(\varphi) : && \\
& \quad p \neq start, \mathbf{TRUE}(T) \rightarrow \mathbf{OCC}(p, T) \vee \mathbf{NOT_OCC}(p, T). && (2V)\\
& \mathbf{OCC}(X, T) \wedge \mathbf{NOT_OCC}(X, T) \rightarrow \bot. && (C)\\
& H(X, T) \wedge X \in \mathcal{V} \rightarrow \mathbf{OCC}(X, T). && (O)\\
& \mathbf{OCC}(X, T) \wedge X \in \mathcal{V} \rightarrow \mathbf{E}(X, T). && (E_1)\\
& \mathbf{NOT_OCC}(X, T) \wedge X \in \mathcal{V} \rightarrow \mathbf{EN}(X, T). \} && (E_2)
\end{aligned}$$

The role of set $\mathcal{V}$, used in the last three constraints, is related to the snf function specified in Definition (5.2). snf introduces a number of auxiliary propositional symbols (for renaming purposes). Intuitively, the translation should not impose expectations on renaming symbols, i.e., it should be restricted to the set $\mathcal{V}$ of events only. In Lemma (5.12) we will restrict the projection to the set $\mathcal{E}\,(\mathtt{snf}\,[\varphi])$. Concerning integrity constraints, $\mathcal{S}$-mapping applies $\mathcal{IC}$-mapping and then augments the obtained constraints with further general rules. Such rules capture specific aspects of the LTL semantics. In particular:

- (S) translates the special **START** symbol, which is introduced by SNF and is true only at the initial state (i.e., at time point 0).
- (T_1) and (T_2) formalize the LTL *true* atom, which is implicitly subject to the formula $\Box(true)$. To this aim, the **TRUE** abducible is introduced, using an initial rule (T_1) and a recursive rule (T_2) to represent that it holds at all time points.
- $(2V)$ and (C) are used to model the two-valued semantics of LTL, i.e., that in each state a proposition is either true or false. Note that $(2V)$ indeed is a shortcut for a set of integrity constraints, a constraint for each propositional symbol $p \in \mathcal{P}(\varphi)$. We exclude the symbol "start", which is introduced by Fisher et al. as a special symbol holding only in the initial state.

- (O), (E_1) and (E_2) relate the (non) occurrence of each proposition in each state with the SCIFF concepts of happened events and positive (negative) expectations. Notice also that for readability reasons we use the expression $X \in \mathcal{V}$: such notation is not allowed in the SCIFF syntax, and it would be substituted by a prolog membership predicate.

The following theorem states that sm preserves compliance, hence an arbitrary SNF formula is translatable to a *behaviourally equivalent* SCIFF specification.

Theorem 5.7. (SCIFF can express SNF formulae [38, p.103])
Given an SNF formula σ and the SCIFF specification $\mathcal{S} = \mathtt{sm}\,[\sigma, \mathcal{P}(\sigma)]$, it holds that $\sigma \overset{c}{\leftrightsquigarrow} \mathcal{S}$.

Proof:
First of all, it is worth noting that LTL and SCIFF share the same semantics for basic logical connectives AND($\wedge$), OR ($\vee$), and implication ($\Rightarrow$ in LTL and $\rightarrow$ in SCIFF). Negation is intended as Negation As Failure. We will therefore focus only on the simplest SNF-forms, consisting of single proposition symbols instead of conjunctions/disjunctions of them. We consider each basic SNF-form separately.

$\sigma = (\mathbf{START} \Rightarrow l)$

If l is a positive literal, say, $l = a$, each compliant LTL execution trace $\mathcal{T}_{\text{LTL}}$ must satisfy the property that $a \in \mathcal{T}_{\text{LTL}}(0)$, because **START** always holds in state 0. The obtained $\mathcal{S}$ contains the corresponding IC

$$\mathtt{icm}[\mathbf{START} \Rightarrow a] = \mathbf{OCC}(start, 0) \rightarrow \mathbf{OCC}(a, 0).$$

By taking into account also the two general ICs (S) and (E_1), all abductive explanations of $\mathcal{S}$ must expect a at time point 0, i.e., they must contain $\mathbf{E}(a, 0)$. Therefore, each compliant trace $\mathcal{T}_{\text{SCIFF}}$ must contain $H(a, 0)$. By considering the trace mapping function tm, this is exactly the same property required for compliant LTL traces, and therefore compliance is preserved by switching from σ to $\mathcal{S}$ or vice-versa. The case in which l is a negative literal, say, $l = \neg a$, can be proven in a similar way: each compliant LTL trace $\mathcal{T}_{\text{LTL}}$ must satisfy the property that $a \notin \mathcal{T}_{\text{LTL}}(0)$, each compliant SCIFF trace $\mathcal{T}_{\text{SCIFF}}$ must satisfy the property that $H(a, 0) \notin \mathcal{T}_{\text{SCIFF}}$, and the two properties are equivalent.

$\sigma = \Box(k \Rightarrow \bigcirc l)$

Let us consider a first case where both k and l are positive literals, and focus on one side of the equivalence ($\overset{c}{\rightsquigarrow}$); the other side can be proven in a very similar way. Towards a contradiction, suppose there exists an execution trace $\mathcal{T}_{\text{LTL}}$ which is compliant with σ, but whose corresponding trace $\mathcal{T}_{\text{SCIFF}}$ is not compliant with $\mathcal{S} = \mathtt{sm}\,[\sigma, \mathcal{P}(\sigma)]$. Notice that, by Definition 5.6 (applying (O) and (E_1)), $\mathcal{S}$ explicitly foresees that in case k happens at a time t, then l is expected to happen at time $t_2, t_2 = t+1$. Hence, to violate $\mathcal{S}$, $\mathcal{T}_{\text{SCIFF}}$ must contain, for a certain time t the event $H(k, t)$, while $H(l, t_2) \notin \mathcal{T}_{\text{SCIFF}}$. By applying the $\mathtt{tm}^{-1}$ function on this trace, one obtains a $\mathcal{T}_{\text{LTL}}$ which obeys the following properties: (1) $k \in \mathcal{T}_{\text{LTL}}(t)$, and (2) $l \notin \mathcal{T}_{\text{LTL}}(t+1)$. The second property in particular implies that $\mathcal{T}_{\text{LTL}}$ is not compliant with σ, hence the initial hypothesis

does not hold. The other side of the implication ($\overset{c}{\leftsquigarrow}$) can be proved in the same way, exploiting again the characteristics of the `tm` function. This same proving schema can be applied also to the case where k is a positive literal, and l is a negative literal: the only difference is that $\mathcal{S}$ will contain a negative expectation **EN**, rather than a positive one as before.

Let us now consider the case in which k is a negative literal, say $k = \neg a$, and l is a positive literal, say $l = b$; again, the case in which l is a negative literal can be proven in the same way. Each compliant $\mathcal{T}_{\text{LTL}}$ trace must obey the following property: $\forall\, t,\ a \in \mathcal{T}_{\text{LTL}}(t) \vee b \in \mathcal{T}_{\text{LTL}}(t+1)$. The IC obtained by the application of `icm` is $\mathbf{NOT\_OCC}(a, T) \rightarrow \mathbf{OCC}(b, T_2) \wedge T_2 = T + 1$. For each time t, if a happens at time t then rule (O) states that $\mathbf{OCC}(a, t)$ is abduced, rule (C) prevents $\mathbf{NOT\_OCC}(a, t)$ to be abduced and thus the IC does not trigger. If, conversely, a does not happen at time t, by rule $(2V)$ we can have two options. In the first, $\mathbf{OCC}(a, t)$ is abduced, which imposes that also $\mathbf{E}(a, t)$ is abduced (rule E_1); since a does not happen at time t, this assumption is not fulfilled. In the second, $\mathbf{NOT\_OCC}(a, t)$ is abduced, the IC triggers, abducing $\mathbf{OCC}(b, t + 1)$, which in turn triggers (E_1), imposing that b is expected to happen at time $t + 1$. Therefore, each SCIFF compliant execution trace $\mathcal{T}_{\text{SCIFF}}$ must satisfy that $\forall\, t, H(a, t) \in \mathcal{T}_{\text{SCIFF}} \vee H(b, t + 1) \in \mathcal{T}_{\text{SCIFF}}$, which is equivalent, under `tm`, to the property on LTL traces.

$\sigma = \Box(k \Rightarrow \Diamond l)$

This case trivially follows from the discussion made for the previous LTL-clause. The only difference is that the constraint $T_2 = T + 1$ is substituted by $T_2 \geq T$ in this more general case.

Having proven that `sm` preserves compliance for each SNF basic form, we must prove that the translation preserves compliance when applied to a conjunction of these forms. This is straightforward, because a trace complies with a SCIFF specification if *all* the integrity constraints are respected. □

5.3. Translation of arbitrary LTL formulae to SCIFF

By exploiting one of the main results presented in [27], and the fact that `sm` is able to represent all the SNF formulae in SCIFF, we now demonstrate that also an arbitrary LTL formula can be encoded in SCIFF preserving compliance. The main technical problem that must be still tackled is the fact that the SNF translation introduces new symbols (used for renaming complex sub-formulae) which do not represent events. At the SNF level, the distinction between concrete events and renaming symbols gets lost, and therefore the SCIFF specification produced by applying in cascade the SNF and the `sm` translation does not preserve compliance with respect to the original LTL formula: positive expectations are imposed also on renaming symbols, which however do not appear in the original LTL formula, leading to a mismatch between the two notions of compliance.

Example 5.8. (Compliance mismatch)
Let $\varphi = \Diamond a$; its SNF translation is $\sigma = \texttt{snf}(\Diamond a) = \Box(\mathbf{START} \Rightarrow \mathbf{Y}) \wedge \Box(\mathbf{Y} \Rightarrow \Diamond a)$. The LTL execution trace containing only the execution of a in state 0 is compliant with φ, but it is not compliant with σ, which requires also the presence of $\mathbf{Y}$ in state 0.

To overcome this issue, the intuitive idea is to propose a translation that does not impose expectations on renaming symbols, i.e., restricts the set $\mathcal{V}$ involved in the definition of the sm function only to events. The (compliant) execution traces considered by the SNF translation and the corresponding SCIFF representation extend the execution traces compliant with the original LTL formula with symbols taken from the renaming set. The first step is therefore to define, in both settings, a suitable trace projection, which filters an execution trace by maintaing only certain symbols (in particular, the ones which correspond to events), ruling out the others.

Definition 5.9. (SCIFF trace projection)
Given a SCIFF execution trace $\mathcal{T}_{\text{SCIFF}}$ and a set $\mathcal{V}$ of predicate symbols, the *trace projection* of $\mathcal{T}_{\text{SCIFF}}$ on $\mathcal{V}$ ($\mathcal{T}_{\text{SCIFF}}|_{\mathcal{V}}$) is the subset of $\mathcal{T}_{\text{SCIFF}}$ containing only events taken from $\mathcal{V}$:

$$\mathcal{T}_{\text{SCIFF}}|_{\mathcal{V}} \triangleq \{H(e,t) \mid H(e,t) \in \mathcal{T}_{\text{SCIFF}} \wedge e \in \mathcal{V}\}$$

Definition 5.10. (LTL trace projection)
Given an LTL execution trace $\mathcal{T}_{\text{LTL}} = (\mathbb{N}, <, v_{occ})$ and a set $\mathcal{V}$ of proposition symbols, the *trace projection* of $\mathcal{T}_{\text{LTL}}$ on $\mathcal{V}$ ($\mathcal{T}_{\text{LTL}}|_{\mathcal{V}}$) is the subset of $\mathcal{T}_{\text{LTL}}$ containing only events taken from $\mathcal{V}$:

$$\mathcal{T}_{\text{LTL}}|_{\mathcal{V}} = (\mathbb{N}, <, {v_{occ}}') \text{ s.t. } {v_{occ}}'(e) \triangleq \begin{cases} v_{occ}(e) & \text{if } e \in \mathcal{V}; \\ \emptyset & \text{otherwise.} \end{cases}$$

Lemma 5.11. (Commutativity between trace projection and trace mapping)
For each LTL execution trace $\mathcal{T}_{\text{LTL}}$ and for each set of proposition symbols $\mathcal{V}$

$$\texttt{tm}\,[\mathcal{T}_{\text{LTL}}|_{\mathcal{V}}] = \texttt{tm}\,[\mathcal{T}_{\text{LTL}}]\,|_{\mathcal{V}}$$

Proof:
From the definitions of trace mapping (Def. 4.1) and of trace projection (Def. 5.9 and 5.10). In particular, let us first consider an element e belonging to $\mathcal{V}$. On the left side, the valuation function of e is maintained by the LTL projection and then subject to the tm mapping; on the right side, the valuation function of e is subject to the tm mapping, and the obtained result is maintained by the SCIFF projection. Let us now consider an element e' outside $\mathcal{V}$. On the left side, the valuation function of e' is "emptied" by the LTL projection, and therefore no happened event concerning e' is produced by tm; on the right side, a set of happened events concerning e' is produced by tm, but they are then ruled out by the SCIFF projection. □

We now briefly recall one of the main results presented in [27], which proves that SNF preserves satisfiability, i.e., in our setting, that it preserves compliance. Lemma 5.12 reviews the satisfiability result by explicitly taking into account execution traces. In particular, it states that execution traces compliant respectively with an LTL formula and its corresponding SNF are exactly the same if we restrict the comparison only to concrete events (without considering the renaming symbols).

Lemma 5.12. (Compliance preservation via extended traces, adapted from [27])
For each LTL formula φ, it holds that

$$\forall\, \mathcal{T}_{\text{LTL}}\ \text{CMP}_{\text{LTL}}\,(\mathcal{T}_{\text{LTL}}, \texttt{snf}\,[\varphi]) \Longrightarrow \text{CMP}_{\text{LTL}}\,\left(\mathcal{T}_{\text{LTL}}|_{\mathbb{E}(\texttt{snf}[\varphi])}, \varphi\right)$$

$$\forall\, \mathcal{T}_{\text{LTL}}\ \text{CMP}_{\text{LTL}}\left(\mathcal{T}_{\text{LTL}}, \varphi\right) \Longrightarrow \exists \mathcal{T}'_{\text{LTL}}\ s.t.\ \mathcal{T}_{\text{LTL}} = \mathcal{T}'_{\text{LTL}}|_{\mathcal{E}(\texttt{snf}[\varphi])} \wedge \text{CMP}_{\text{LTL}}\left(\mathcal{T}'_{\text{LTL}}, \texttt{snf}\left[\varphi\right]\right)$$

where we remember that (by Definition 5.3) $\mathcal{E}\left(\texttt{snf}\left[\varphi\right]\right) = \mathcal{P}(\varphi)$.

Theorem 5.13. (SNF preserves satisfiability, adapted from [27])
An LTL formula φ is satisfiable (i.e., $\exists \mathcal{T}_{\text{LTL}}, \exists i \in \mathbb{N}$ s.t. $\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \varphi$) iff $\texttt{snf}(\varphi)$ is satisfiable.

With such preliminaries, it is possible to prove that each LTL formula is translatable to a SCIFF specification, preserving compliance.

Theorem 5.14. (SCIFF can express LTL)
Given an arbitrary LTL formula φ and the SCIFF specification $\mathcal{S} = \texttt{sm}\left[\texttt{snf}\left[\varphi\right], \mathcal{P}(\varphi)\right]$, it holds that $\mathcal{S} \overset{c}{\leftrightarrows} \varphi$.

Proof:
Let us denote $\sigma = \texttt{snf}\left[\varphi\right]$. From Def. 4.3, and by remembering that the event set of σ contains all the proposition symbols of φ ($\mathcal{P}(\varphi) = \mathcal{E}(\sigma)$), one has to prove that for each $\mathcal{T}_{\text{LTL}}$it holds

$$\text{CMP}_{\text{LTL}}(\mathcal{T}_{\text{LTL}}, \varphi) \Longleftrightarrow \text{CMP}_{\text{SCIFF}}\left(\texttt{tm}\left[\mathcal{T}_{\text{LTL}}\right], \texttt{sm}\left[\sigma, \mathcal{E}(\sigma)\right]\right)$$

Firstly we will prove one way of the implication ($\Longrightarrow$), and then the opposite direction ($\Longleftarrow$). Both proofs are organized in the same way: by applying the results obtained in Lemma 5.11, Lemma 5.12, and Theorem 5.7, the problem of proving a formula is reduced to proving another, simpler formula. Hence, each proof starts with a diagram that shows how each previous result is applied to a formula, and then the simpler formula is proved.
($\Longrightarrow$) Let us consider the following schema, where for all $\mathcal{T}_{\text{LTL}}$:

$$\begin{array}{ccc}
\text{CMP}_{\text{LTL}}(\mathcal{T}_{\text{LTL}}, \varphi) & \xrightarrow{\quad(*)\quad} & \text{CMP}_{\text{SCIFF}}\left(\texttt{tm}\left[\mathcal{T}_{\text{LTL}}\right], \texttt{sm}\left[\sigma, \mathcal{E}(\sigma)\right]\right) \\
\Big\Downarrow \text{Lemma 5.12} & & \Big\Uparrow (\dagger) \\
\exists \mathcal{T}'_{\text{LTL}} s.t. \left\{\mathcal{T}_{\text{LTL}} = \mathcal{T}'_{\text{LTL}}|_{\mathcal{E}(\sigma)} \wedge \text{CMP}_{\text{LTL}}(\mathcal{T}'_{\text{LTL}}, \sigma)\right\} & \xrightarrow{\quad\text{Theorem 5.7}\quad} & \text{CMP}_{\text{SCIFF}}\left(\texttt{tm}\left[\mathcal{T}'_{\text{LTL}}\right], \texttt{sm}\left[\sigma, \mathcal{P}(\sigma)\right]\right)
\end{array}$$

The schema shows that proving $(*)$ reduces to proving (†), i.e., to proving that

$$\text{CMP}_{\text{SCIFF}}\left(\texttt{tm}\left[\mathcal{T}'_{\text{LTL}}\right], \texttt{sm}\left[\sigma, \mathcal{P}(\sigma)\right]\right) \Longrightarrow \text{CMP}_{\text{SCIFF}}\left(\texttt{tm}\left[\mathcal{T}'_{\text{LTL}}|_{\mathcal{E}(\sigma)}\right], \texttt{sm}\left[\sigma, \mathcal{E}(\sigma)\right]\right) \qquad (\dagger)$$

By taking into account abducible sets, Def. 5.3 and Lemma 5.11, (†) becomes:

$$\text{CMP}_{\text{SCIFF}}{}^{\Delta}\left(\mathcal{T}_{\text{SCIFF}}, \mathcal{S}^{\mathcal{ER}}\right) \Longrightarrow \text{CMP}_{\text{SCIFF}}{}^{\Delta'}\left(\mathcal{T}_{\text{SCIFF}}|_{\mathcal{E}(\sigma)}, \mathcal{S}^{\mathcal{E}}\right) \qquad (\ddagger)$$

where $\mathcal{S}^{\mathcal{ER}} = \texttt{sm}\left[\sigma, \mathcal{E}(\sigma) \cup \mathcal{R}(\sigma)\right]$, $\mathcal{S}^{\mathcal{E}} = \texttt{sm}\left[\sigma, \mathcal{E}(\sigma)\right]$ and $\mathcal{T}_{\text{SCIFF}} = \texttt{tm}\left[\mathcal{T}'_{\text{LTL}}\right]$. To prove (‡), we demonstrate that

$$\Delta' = \Delta \setminus \left(\{\mathbf{E}(e,t) | e \in \mathcal{R}(\sigma)\} \cup \{\mathbf{EN}(e,t) | e \in \mathcal{R}(\sigma)\}\right)$$

obeys the three properties required by the Definition 3.8 of SCIFF compliance:

1. Δ' is an abductive explanation for $\mathcal{S}^{\mathcal{E}}_{\mathcal{T}_{\text{SCIFF}}|_{\mathcal{E}(\sigma)}}$. The only difference between $\mathcal{S}^{\mathcal{E}}$ and $\mathcal{S}^{\mathcal{ER}}$ is that, for the first specification, rules (O), (E_1) and (E_2) of Def. 5.6 do not trigger for events outside $\mathcal{E}(\sigma)$ (in particular, they do not trigger for events inside $\mathcal{R}(\sigma)$). From Remark 3.4 in Section 3.2, Δ is therefore a suitable abductive explanation for $\mathcal{S}^{\mathcal{E}}$ too. Furthermore, being (E_1) and (E_2) the only constraints involving positive and negative expectations concerning elements in $\mathcal{R}(\sigma)$, it is not required for an abductive explanation to contain them anymore.

2. Δ' is **E**-consistent, because $\Delta' \subseteq \Delta$ and Δ is **E**-consistent.

3. Δ' is $\mathcal{T}_{\text{SCIFF}}|_{\mathcal{E}(\sigma)}$-fulfilled. Since $\mathcal{T}_{\text{SCIFF}}|_{\mathcal{E}(\sigma)}$ is a projection of $\mathcal{T}_{\text{SCIFF}}$, $\Delta' \subseteq \Delta$ and Δ is $\mathcal{T}_{\text{SCIFF}}$-fulfilled, no negative expectation in Δ' can be violated by $\mathcal{T}_{\text{SCIFF}}|_{\mathcal{E}(\sigma)}$. Positive expectations concerning elements in $\mathcal{E}(\sigma)$ are maintained in Δ', and so are the corresponding happened events after the trace projection. Positive expectations concerning elements in $\mathcal{R}(\sigma)$ are removed from Δ when obtaining Δ', and therefore the application of the trace projection, which rules out happened events concerning elements in $\mathcal{R}(\sigma)$, does not compromise trace fulfillment.

($\Longleftarrow$) We move then to prove the reverse direction of the double implication stated in this theorem. Again, let us consider the following schema, where for all $\mathcal{T}_{\text{SCIFF}}$ it holds:

$$\begin{array}{ccc}
\text{CMP}_{\text{LTL}}\left(\mathtt{tm}^{-1}\left[\mathcal{T}_{\text{SCIFF}}\right], \varphi\right) & \xleftarrow{\quad (**) \quad} & \text{CMP}_{\text{SCIFF}}\left(\mathcal{T}_{\text{SCIFF}}, \mathtt{sm}\left[\sigma, \mathcal{E}(\sigma)\right]\right) \\
\Big\Uparrow \text{Lemma 5.12, then Lemma 5.11} & & \Big\Downarrow (\S) \\
\text{CMP}_{\text{LTL}}\left(\mathtt{tm}^{-1}\left[\mathcal{T}'_{\text{SCIFF}}\right], \sigma\right) & \xleftarrow{\text{Theorem 5.7}} & \begin{array}{l}\exists\, \mathcal{T}'_{\text{SCIFF}} \text{ s.t. } \left\{\mathcal{T}_{\text{SCIFF}} = \mathcal{T}'_{\text{SCIFF}}|_{\mathcal{E}(\sigma)}\right. \\ \left. \wedge \text{CMP}_{\text{SCIFF}}\left(\mathcal{T}'_{\text{SCIFF}}, \mathtt{sm}\left[\sigma, \mathcal{P}(\sigma)\right]\right)\right\}\end{array}
\end{array}$$

The schema shows that proving $(**)$ reduces to proving $(\S)$, i.e. to proving that

$$\forall\, \mathcal{T}_{\text{SCIFF}},\ \text{CMP}_{\text{SCIFF}}{}^{\Delta}\left(\mathcal{T}_{\text{SCIFF}}, \mathcal{S}^{\mathcal{E}}\right) \Longrightarrow \exists\, \mathcal{T}'_{\text{SCIFF}},\ \mathcal{T}_{\text{SCIFF}} = \mathcal{T}'_{\text{SCIFF}}|_{\mathcal{E}(\sigma)} \wedge \text{CMP}_{\text{SCIFF}}{}^{\Delta'}\left(\mathcal{T}'_{\text{SCIFF}}, \mathcal{S}^{\mathcal{ER}}\right) \tag{$\S$}$$

where $\mathcal{S}^{\mathcal{ER}} = \mathtt{sm}\left[\sigma, \mathcal{E}(\sigma) \cup \mathcal{R}(\sigma)\right]$ and $\mathcal{S}^{\mathcal{E}} = \mathtt{sm}\left[\sigma, \mathcal{E}(\sigma)\right]$.

First of all, it is worth noting that $\mathcal{S}^{\mathcal{ER}}$ extends $\mathcal{S}^{\mathcal{E}}$ by imposing that rules (O), (E_1) and (E_2) can be also triggered by **OCC/NOT_OCC** abducibles involving symbols in $\mathcal{R}(\sigma)$, generating a larger set of expectations. Since $\mathcal{T}'_{\text{SCIFF}} \supseteq \mathcal{T}_{\text{SCIFF}}$, an abductive explanation Δ' can be therefore found for $\mathcal{S}^{\mathcal{ER}}$ by extending Δ with the new generated expectations: $\Delta' = \Delta \cup \Delta^{\mathbf{E}}_{\mathcal{R}} \cup \Delta^{\mathbf{EN}}_{\mathcal{R}}$, where $\Delta^{\mathbf{E}}_{\mathcal{R}}$ and $\Delta^{\mathbf{EN}}_{\mathcal{R}}$ respectively represent the inserted positive and negative expectations.

Δ' is **E**-consistent. Indeed, since $\Delta^{\mathbf{E}}_{\mathcal{R}}$ and $\Delta^{\mathbf{EN}}_{\mathcal{R}}$ contain only expectations generated by rules (E_1) and (E_2), by construction we have:

$$\begin{array}{l}
\forall\, \mathbf{E}(a,t), \qquad \mathbf{E}(a,t) \in \Delta^{\mathbf{E}}_{\mathcal{R}} \Rightarrow \mathbf{OCC}(a,t) \in \Delta' \\
\forall\, \mathbf{EN}(a,t),\ \mathbf{EN}(a,t) \in \Delta^{\mathbf{EN}}_{\mathcal{R}} \Rightarrow \mathbf{NOT_OCC}(a,t) \in \Delta'
\end{array} \tag{$\S\S$}$$

Towards a contradiction, assume there exist a, t (with $a \in \mathcal{R}(\sigma)$) s.t. $\mathbf{E}(a,t) \in \Delta^{\mathbf{E}}_{\mathcal{R}}$ and $\mathbf{EN}(a,t) \in \Delta^{\mathbf{EN}}_{\mathcal{R}}$. In this case, (§§) would state that $\mathbf{OCC}(a,t) \in \Delta'$ and $\mathbf{NOT\_OCC}(a,t) \in \Delta'$. This would violate rule (C).

Therefore Δ' is an abductive explanation, and we can construct an execution trace $\mathcal{T}'_{\text{SCIFF}}$ compliant with $\mathcal{S}^{\mathcal{ER}}$ as follows:

$$\mathcal{T}'_{\text{SCIFF}} = \mathcal{T}_{\text{SCIFF}} \cup \mathcal{T}^{\mathcal{R}}_{\text{SCIFF}}, \text{ where } H(a,t) \in \mathcal{T}^{\mathcal{R}}_{\text{SCIFF}} \Leftrightarrow \mathbf{E}(a,t) \in \Delta^{\mathbf{E}}_{\mathcal{R}}$$

Notice that:

1. Δ' is left untouched by $\mathcal{T}'_{\text{SCIFF}}$. Indeed, the only impact of $\mathcal{T}^{\mathcal{R}}_{\text{SCIFF}}$ on the ICs of $\mathcal{S}^{\mathcal{ER}}$ is to trigger rule (O), generating corresponding $\mathbf{OCC}$ abducibles. However, from (§§) we know that all these abducibles are already contained in Δ'.

2. Δ' is $\mathcal{T}'_{\text{SCIFF}}$-fulfilled by construction.

3. $\mathcal{T}'_{\text{SCIFF}}|_{\mathcal{E}(\sigma)} = \mathcal{T}_{\text{SCIFF}}$, because all the happened events contained in $\mathcal{T}^{\mathcal{R}}_{\text{SCIFF}}$ involve symbols belonging to $\mathcal{R}(\sigma)$, and are therefore ruled out by applying the projection.

□

6. Discussion and related works

In this work we compare the framework SCIFF with the widely adopted LTL, from the viewpoint of the compliance verification task. To this end, we have proposed a formal notion of compliance for each one of the approaches, and defined the equivalence of the two notions. Then, exploiting a previous result in [27] about Separated Normal Form of LTL formulae, we provide an automatic translation between LTL-based and SCIFF-based specifications. We formally prove that such translation indeed preserves the notion of compliance, with respect to the previously defined equivalence.

Several research works have focused on the translation of temporal logics specifications into logic programs, with the aim of checking temporal logic properties. Among them, we recall that ASP was applied to the problem of bounded model checking in Petri nets [39]. The proposed encoding models associates each step in the monitored net with an integer number. Since ASP solving is based on a bottom-up computation of the models, it relies on a grounding phase of the program, in which the objective is to obtain a (subset of the) complete grounding that must be stored in memory. This limits the application of the encoding to bounded traces of events. Nevertheless, the experimental results show that ASP is a quite competitive approach to the problem of bounded model checking with respect to the NuSMV model checker. The main difference of our work with [39] is that we do not approach the model checking problem, but we encode LTL into abductive logic programming for the problem of compliance verification.

Delzanno and Podelski [40] propose to translate a concurrent system specification S into a CLP program P_S. Computational Tree Logic (CTL) properties, as existential properties, are then represented with the set of states satisfying each of them. These sets are obtained by composing (according to two operators) sets of constrained facts F (a.k.a. of intentional representation of sets of atoms)

with the translated program, and by the least and the greatest fix-point sets of these compositions. Universal properties can be checked in the complement of the two fix-point sets. They therefore adopt a bottom-up approach, based on fix-points, to represent CTL properties. Nonetheless, the temporal properties are undecidable for the general class of concurrent systems considered by Delzanno and Podelski (as in our case). Therefore, for checking CTL properties, the authors propose a possibly non-terminating model-checking procedure. They introduce abstractions in it to enforce or (simply speed-up) termination on practical examples. An implementation in SICStus Prolog is also provided.

In [41] the XSB logic programming language that extends Prolog with tabled resolution is exploited to encode in a declarative way CCS-like languages and a fragment of the modal mu-calculus, and to perform model checking. The good performances, thanks to the use of XSB and optimization techniques, show the feasibility of the approach. We share with [41] the use of logic programming techniques, but we focus on the preservation of the notion of compliance through abduction and the SCIFF framework, while in [41] the aim is to implement (trough XSB Prolog) a model checking algorithm for verification. Exploiting optimization techniques in SCIFF for making the encoding efficient and effective are for us subject of future work.

Nilsson and Lübcke [42] propose a model checking approach for a semantically complete fragment of CTL based on CLP, tabling, and a limited form of constructive negation. They address both symbolic model checking and local model checking. Since no implementation was available of tabling with CLP and constructive negation, they developed their own implementation, not as a general tabling system, but by hard coding tabling into the model checker. Preliminary experiments on the example of the dining philosophers show that the approach is competitive in some cases with local and global model checkers. The differences with our approach are mainly that Nilsson and Lübcke use CTL instead of LTL, and that we do not address a model checking problem, but we translate LTL into a SCIFF specification to address compliance verification. Interestingly, SCIFF also encloses constraints and constructive negation, but it does not use tabling.

In [43] the addressed application domain is the same of Business Processes, with the aim of modelling procedural and declarative (semantic) knowledge. An LP approach, and in particular the Fluent Calculus, is used for modelling and executing different reasoning tasks such as verification and trace compliance. Specifically, a model checking methodology based on CTL is implemented by means of logic programming rules. Our work is related to [43] in advocating the importance of translating relevant aspects of workflow in an expressive language logic-based. Distinctive in our approach is the use of Abductive Logic Programming instead of the fluent calculus. Moreover, we are not interested here to mimic model checking techniques, but rather to investigate if and how models are preserved from one logic to another.

The Business Process domain is addressed also in [44], where the authors exploit the Dynamic Linear Time Temporal Logic (DLTL) [45], extended with constraints upon variables with finite domains. The resulting language is essentially a propositional language and preserves the decidability property of DLTL. Authors borrow from previous works a translation of the BP domain to ASP, together with an ASP encoding of Bounded Model Checking (BMC). The emphasis is about verification of temporal properties of a business process, including verification of compliance to business rules. In a previous work [21] we focused on the same issue, but ALP was used instead of ASP, and the SCIFF proof procedure was exploited to reasoning on a small LTL fragment.

The exploitation of LTL Separated Normal Form formulae raised some technical problems. SNF allows us to prove that *any* LTL formula can be translated into a SCIFF specification while preserving behavioural equivalence. Unfortunately, the translation of an LTL formula into its SNF equivalent comes with the introduction of several additional propositional symbols (for renaming complex subformulae). Section 5.3 is entirely devoted to prove that, despite the introduction of these renaming symbols, the behavioural equivalence with respect to compliance is preserved. In some previous works [20, 8] we took a different approach: by focusing on a significant fragment of LTL, we were able to provide ad-hoc translation into a SCIFF specification. However, the price to be paid was the loss of generality with respect to the approach presented here.

7. Conclusions

LTL-based techniques for verification have a number of strengths and weaknesses, as well as the SCIFF framework that inherits advantages and limits of the logic programming approaches. An important result of this work is to better clarify the links between the two techniques: this opens up to the possibility of an integrated approach based on First Order Logic and Computational Logic in particular, where the best of both worlds (LTL and SCIFF) can be exploited. Indeed, ALP can be actually exploited for throwing a bridge between systems using LTL for dealing with compliance issues and trace verification (such as [6, 10, 46]), and the ones based on (extensions of) logic programming (see e.g. [47, 16, 18, 21]). Moreover, under specific conditions, SCIFF can be used to prove the satisfiability of properties, as discussed in [21]. Finally, a further advantage is given by the fact that a number of model checker tools are available for LTL; SCIFF as well is equipped with a proof procedure[9]. Users can focus on a logic formula expressing the property to be verified; they can express such formula in LTL, and then they can freely choose the best tool depending on the task: e.g., SCIFF proof procedure might be preferable when dealing at run-time with dynamic, open, evolving environments; model checkers might be used instead to verify traces of infinite length.

A limit of the presented approach stems from the semi-decidability issues of (refutation-based) logic programming, and by the discussed automatic translation technique of LTL specifications into SCIFF specifications. The SCIFF framework suffers the same semi-decidability issues of LP: SCIFF ensures termination [16] by imposing syntactic restrictions on the logic programs and integrity constraints, and in particular, acyclicity and boundedness conditions of abductive logic programs [48]. However, the automatic translation presented here does not meet these restrictions: in particular boundedness is not guaranteed, due to the presence of infinite traces and of infinite abductive explanations. In particular, the proposed translation of SNF *step LTL clauses* into SCIFF introduces an infinite number of abductive explanations. Fisher and colleagues pointed out in [49] that "mechanising modal (or temporal) logics becomes complex when interactions occur between modalities $\bigcirc$ (next time) and $\Box$ (always), since the interaction is of an *inductive* nature, which can be particularly complex". In the past many attempts have been done to overcome the problem: for example, the authors in [50] propose a CLP-based model checking technique, where the model of the observed system is specified through

[9] SCIFF Web Site: `http://lia.disi.unibo.it/sciff/`

an automaton. However, they cannot handle infinite sequences without the intervention of the user specifying a finite number of event sequences.

From a practical viewpoint the problem can be avoided by restricting to a significant fragment of LTL, and providing ad-hoc translations for which termination is guaranteed. For example, in [20, 8] we investigated how declarative business constraints (whose semantics was given in terms of a small LTL fragment) could be represented in SCIFF and, more important, could be reasoned upon. In [28] we compared SCIFF performances against model checking techniques, showing that logic programming and abduction can be (under specific conditions) a viable alternative to LTL techniques for compliance verification.

Alternatively, it is possible to notice that a number of applications inherently require finite traces, like e.g. business processes [6], that are developed to reach a business goal (such as delivery of a product) in a finite number of finite steps. Notice also that, as argued in [51], a huge class of LTL patterns are insensitive to "infiniteness", i.e., their interpretation over finite traces is obtained from the standard infinite-trace semantics by just adding a further proposition "end" that becomes true at some point and that makes it impossible for any other proposition to be true (this does not hold in general). For such a class of properties, the translation presented here can be directly employed in a finite-trace setting, making it possible to readily apply the SCIFF proof procedure for monitoring purposes. In general, the finite trace assumption can be done for domains in which LTL formulae are used for monitoring and run-time verification, because even if the overall trace of the system could be infinite, reasoning must be carried out on partial, evolving finite traces [46]. Hence, by adopting a finite trace semantics for LTL (such as in [52]), SCIFF can be exploited for monitoring LTL specifications without incurring into the termination issue. Indeed, the SCIFF declarative semantics is specifically thought to deal with dynamic, evolving environments, and can have both a closed or an open flavour, capturing in this way finite as well as open (i.e., subject to extension) sequences of events. Automatic translation of any LTL formula within a finite trace semantics into a SCIFF corresponding model is matter of ongoing work.

Acknoledgements. The authors would like to thank the reviewers, whose comments and suggestions greatly helped to improve the quality of this work.

This research has been partially supported by the Euregio IPN12 *"KAOS: Knowledge-Aware Operational Support"* project, which is funded by the "European Region Tyrol-South Tyrol-Trentino" (EGTC) under the first call for basic research projects, by the UNIBZ internal project *KENDO (Knowledge-driven ENterprise Distributed cOmputing)*, and by GNCS project DECORE.

References

[1] Prior AN. Past, Present and Future. Oxford University Press, 1967.

[2] Emerson EA. Temporal and Modal Logic. In: van Leeuwen J (ed.), Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics. Elsevier and MIT Press. 1990. ISBN: 0-444-88074-7, 0-262-22039-3.

[3] Holzmann G. The model checker SPIN. *Software Engineering, IEEE Transactions on*. 1997;**23**(5):279–295. doi:10.1109/32.588521.

[4] Cimatti A, Clarke EM, Giunchiglia E, Giunchiglia F, Pistore M, Roveri M, Sebastiani R, Tacchella A. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: Brinksma E, Larsen KG (eds.), CAV, volume 2404 of *Lecture Notes in Computer Science*. Springer, 2002 pp. 359–364. ISBN: 3-540-43997-8.

[5] Pesic M, Schonenberg H, van der Aalst WMP. DECLARE: Full Support for Loosely-Structured Processes. In: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007). IEEE Computer Society, 2007 pp. 287–300. doi:10.1109/EDOC.2007.14.

[6] Pesic M, van der Aalst WMP. A Declarative Approach for Flexible Business Processes Management. In: Eder J, Dustdar S (eds.), Proceedings of the BPM 2006 Workshops (BPD, BPI, ENEI, GPWW, DPM, semantics4ws), volume 4103 of *Lecture Notes in Computer Science*. Springer, 2006 pp. 169–180. URL `https://doi.org/10.1007/11837862_18`.

[7] Pesic M. Constraint-Based Workflow Management Systems: Shifting Controls to Users. Ph.D. thesis, Beta Research School for Operations Management and Logistics, Eindhoven, 2008.

[8] Montali M. Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach, volume 56 of *Lecture Notes in Business Information Processing*. Springer, 2010. doi:10.1007/978-3-642-14538-4_1.

[9] Awad A, Decker G, Weske M. Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In: Dumas M, Reichert M, Shan MC (eds.), 6th International Conference on Business Process Management (BPM 2008), volume 5240 of *Lecture Notes in Computer Science*. Springer, 2008 pp. 326–341. URL `https://doi.org/10.1007/978-3-540-85758-7_24`.

[10] van der Aalst W, de Beer H, van Dongen B. Process Mining and Verification of Properties: An Approach based on Temporal Logic. In: Meersman R, Tari Z (eds.), Proceedings of the OTM 2005 Confederated International Conferences CoopIS, DOA, and ODBASE, volume 3760 of *Lecture Notes in Computer Science*. Springer, 2005 pp. 130–147. URL `https://doi.org/10.1007/11575771_11`.

[11] Artikis A, Sergot M, Pitt J. Specifying Norm-Governed Computational Societies. *ACM Transactions on Computational Logic*, 2009;**10**(1):1–42. doi:http://doi.acm.org/10.1145/1459010.1459011.

[12] Fornara N, Colombetti M. Specifying Artificial Institutions in the Event Calculus. In: Dignum [53], 2009 pp. 335–366.

[13] Roman D, Kifer M. Semantic Web Service Choreography: Contracting and Enactment. In: Sheth AP, Staab S, Dean M, Paolucci M, Maynard D, Finin TW, Thirunarayan K (eds.), Proceedings of the 7th International Semantic Web Conference (ISWC 2008), volume 5318 of *Lecture Notes in Computer Science*. Springer, 2008 pp. 550–566. URL `https://doi.org/10.1007/978-3-540-88564-1_35`.

[14] Baldoni M, Baroglio C, Martelli A, Patti V. Reasoning about Interaction Protocols for Customizing Web Service Selection and Composition. *Journal of Logic and Algebraic Programming, special issue on Web Services and Formal Methods*, 2007;**70**(1):53–73. URL `https://doi.org/10.1016/j.jlap.2006.05.005`.

[15] De Nicola A, Missikoff M, Proietti M, Smith F. A Logic-Based Method for Business Process Knowledge Base Management. In: Bergamaschi S, Lodi S, Martoglia R, Sartori C (eds.), 8th Italian Symposium on Advanced Database Systems. Rimini, Italy, 2010 pp. 170–181.

[16] Alberti M, Chesani F, Gavanelli M, Lamma E, Mello P, Torroni P. Verifiable agent interaction in abductive logic programming: the SCIFF framework. *ACM Transactions on Computational Logic*, 2008;**9**(4):29:1–29:43. doi:10.1145/1380572.1380578.

[17] Alberti M, Gavanelli M, Lamma E, Mello P, Torroni P, Sartor G. Mapping deontic operators to abductive expectations. *Computational & Mathematical Organization Theory*, 2006;**12**(2-3):205–225. URL https://doi.org/10.1007/s10588-006-9544-8.

[18] Chesani F, Mello P, Montali M, Torroni P. Commitment Tracking via the Reactive Event Calculus. In: Boutilier C (ed.), Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), 2009 pp. 91–96. URL http://dl.acm.org/citation.cfm?id=1661445.1661461.

[19] Alberti M, Cattafi M, Chesani F, Gavanelli M, Lamma E, Mello P, Montali M, Torroni P. A Computational Logic Application Framework for Service Discovery and Contracting. *Int. J. Web Service Res.*, 2011;**8**(3):1–25. doi:10.4018/IJWSR.2011070101. URL https://doi.org/10.4018/IJWSR.2011070101.

[20] Montali M, Pesic M, van der Aalst WMP, Chesani F, Mello P, Storari S. Declarative Specification and Verification of Service Choreographies. *ACM Transactions on the Web*, 2010;**4**(1):3-1–3-62. doi:10.1145/1658373.1658376.

[21] Montali M, Torroni P, Chesani F, Mello P, Alberti M, Lamma E. Abductive Logic Programming as an Effective Technology for the Static Verification of Declarative Business Processes. *Fundamenta Informaticae*, 2010;**102**(3-4):325–361. URL http://dl.acm.org/citation.cfm?id=1890507.1890512.

[22] Fung TH, Kowalski RA. The IFF Proof Procedure for Abductive Logic Programming. *Logic Programming*, 1997;**33**(2):151–165. URL https://doi.org/10.1016/S0743-1066(97)00026-5.

[23] Dumas M, Rosa ML, Mendling J, Reijers HA. Fundamentals of Business Process Management. Springer, 2013. ISBN: 978-3-642-33142-8. doi:10.1007/978-3-642-33143-5. URL http://dx.doi.org/10.1007/978-3-642-33143-5.

[24] Chopra AK, Singh MP. Producing Compliant Interactions: Conformance, Coverage, and Interoperability. In: Proceedings of the 4th International Conference on Declarative Agent Languages and Technologies, DALT'06. Springer-Verlag, Berlin, Heidelberg, 2006 pp. 1–15. ISBN: 3-540-68959-1, 978-3-540-68959-1. doi:10.1007/11961536_1. URL http://dx.doi.org/10.1007/11961536_1.

[25] Governatori G, Rotolo A. Norm Compliance in Business Process Modeling. In: RuleML 2010, Procs., volume 6403 of *LNCS*. Springer, 2010 pp. 194–209. ISBN: 978-3-642-16288-6.

[26] Governatori G, Hashmi M, Lam H, Villata S, Palmirani M. Semantic Business Process Regulatory Compliance Checking Using LegalRuleML. In: EKAW 2016, Procs., volume 10024 of *LNCS*. Springer, 2016 pp. 746–761. ISBN: 978-3-319-49003-8.

[27] Fisher M, Dixon C, Peim M. Clausal Temporal Resolution. *ACM Transactions on Computational Logic*, 2001;**2**(1):12–56. doi:10.1145/371282.371311.

[28] Montali M, Torroni P, Alberti M, Chesani F, Gavanelli M, Lamma E, Mello P. Verification from Declarative Specifications Using Logic Programming. In: de la Banda MG, Pontelli E (eds.), Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings, volume 5366 of *Lecture Notes in Computer Science*. Springer, 2008 pp. 440–454. ISBN: 978-3-540-89981-5, doi:10.1007/978-3-540-89982-2_39. URL http://dx.doi.org/10.1007/978-3-540-89982-2_39.

[29] Kakas AC, Kowalski RA, Toni F. Abductive Logic Programming. *Journal of Logic and Computation*, 1993;**2**(6):719–770.

[30] Lloyd JW. Foundations of Logic Programming. Springer, 2nd edition, 1987. doi:10.1007/978-3-642-83189-8.

[31] Jaffar J, Maher MJ. Constraint Logic Programming: a Survey. *Logic Programming*, 1994;**19-20**:503–582. URL https://doi.org/10.1016/0743-1066(94)90033-7.

[32] Torroni P, Chesani F, Yolum P, Gavanelli M, Singh MP, Lamma E, Alberti M, Mello P. Modelling Interactions via Commitments and Expectations. In: Dignum [53], 2009 pp. 263–284.

[33] Kunen K. Negation in Logic Programming. *J. Log. Program.*, 1987;**4**(4):289–308. URL https://doi.org/10.1016/0743-1066(87)90007-0.

[34] Clark KL. Negation as Failure. In: Gallaire H, Minker J (eds.), Logic and Data Bases, pp. 293–322. Plenum Press, 1978.

[35] Jaffar J, Maher MJ, Marriott K, Stuckey PJ. The Semantics of Constraint Logic Programs. *J. Log. Program.*, 1998;**37**(1-3):1–46. URL https://doi.org/10.1016/S0743-1066(98)10002-X.

[36] Evans C, Kakas AC. Hypothetico-deductive Reasoning. In: Fifth Generation Computer Systems. IOS Press, 1992 pp. 546–554.

[37] Alberti M, Gavanelli M, Lamma E, Mello P, Torroni P. Abduction with hypotheses confirmation. In: Giunchiglia F (ed.), IJCAI-05 Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence. Professional Book Center, USA, 2005 pp. 1545–1546. ISBN: 0-938075-93-4.

[38] Montali M. Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach. Ph.D. thesis, University of Bologna, 2009.

[39] Heljanko K, Niemelä I. Bounded LTL model checking with stable models. *TPLP*, 2003;**3**(4-5):519–550. doi:10.1017/S1471068403001790. URL http://dx.doi.org/10.1017/S1471068403001790.

[40] Delzanno G, Podelski A. Constraint-based deductive model checking. *STTT*, 2001;**3**(3):250–270. doi:10.1007/s100090100049. URL http://dx.doi.org/10.1007/s100090100049.

[41] Ramakrishna YS, Ramakrishnan CR, Ramakrishnan IV, Smolka SA, Swift T, Warren DS. Efficient Model Checking Using Tabled Resolution. In: Grumberg O (ed.), Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings, volume 1254 of *Lecture Notes in Computer Science*. Springer, 1997 pp. 143–154. ISBN: 3-540-63166-6. doi:10.1007/3-540-63166-6_16. URL http://dx.doi.org/10.1007/3-540-63166-6_16.

[42] Nilsson U, Lübcke J. Constraint Logic Programming for Local and Symbolic Model-Checking. In: Lloyd JW, Dahl V, Furbach U, Kerber M, Lau K, Palamidessi C, Pereira LM, Sagiv Y, Stuckey PJ (eds.), Computational Logic - CL 2000, First International Conference, London, UK, 24-28 July, 2000, Proceedings, volume 1861 of *Lecture Notes in Computer Science*. Springer, 2000 pp. 384–398. ISBN: 3-540-67797-6. doi:10.1007/3-540-44957-4_26. URL http://dx.doi.org/10.1007/3-540-44957-4_26.

[43] Smith F, Proietti M. Rule-based Behavioral Reasoning on Semantic Business Processes. In: Filipe J, Fred ALN (eds.), ICAART 2013 - Proceedings of the 5th International Conference on Agents and Artificial Intelligence, Volume 2, Barcelona, Spain, 15-18 February, 2013. SciTePress, 2013 pp. 130–143. ISBN: 978-989-8565-39-6.

[44] Giordano L, Martelli A, Spiotta M, Dupré DT. Business process verification with constraint temporal answer set programming. *TPLP*, 2013;**13**(4-5):641–655. doi:10.1017/S1471068413000409. URL http://dx.doi.org/10.1017/S1471068413000409.

[45] Henriksen JG, Thiagarajan PS. Dynamic Linear Time Temporal Logic. *Ann. Pure Appl. Logic*, 1999;**96**(1-3):187–207. doi:10.1016/S0168-0072(98)00039-6. URL http://dx.doi.org/10.1016/S0168-0072(98)00039-6.

[46] Havelund K, Rosu G. Testing Linear Temporal Logic Formulae on Finite Execution Traces. Technical Report TR 01-08, RIACS Technical Report, 2001.

[47] Gelfond M, Lobo J. Authorization and Obligation Policies in Dynamic Systems. In: De La Banda MG, Pontelli E (eds.), 24th International Conference on Logic Programming (ICLP), number 5366 in Lecture Notes in Computer Science. Springer, 2008 pp. 22–36. URL https://doi.org/10.1007/978-3-540-89982-2_7

[48] Xanthakos I. Semantic Integration of Information by Abduction. Ph.D. thesis, Imperial College London, 2003.

[49] Fisher M. Implementing Temporal Logics: Tools for Execution and Proof (Tutorial Paper). In: Toni F, Torroni P (eds.), CLIMA VI, volume 3900 of *Lecture Notes in Computer Science*. Springer. 2005 pp. 129–142. ISBN: 3-540-33996-5.

[50] Gupta G, Pontelli E. A constraint-based approach for specification and verification of real-time systems. In: IEEE Real-Time Systems Symposium. IEEE Computer Society, 1997 pp. 230–239. doi:10.1109/REAL.1997.641285.

[51] De Giacomo G, De Masellis R, Montali M. Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness. In: Brodley CE, Stone P (eds.), Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI). AAAI Press/The MIT Press, 2014 pp. 1027–1033. ISBN: 978-1-57735-661-5. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8575.

[52] Bauer A, Leucker M, Schallhart C. Comparing LTL Semantics for Runtime Verification. *Logic and Computation*, 2010;20(3):651–674. doi:10.1093/logcom/exn075.

[53] Dignum V (ed.). Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models. IGI Global, 2009. ISBN: 1605662569.

A. List of the symbols used within the paper

$\mathcal{M}$	An LTL model $\mathcal{M}$(see Definition 2.1)
$\mathcal{T}_{\mathbf{LTL}}$	An LTL execution trace (see Definition 2.2)
v_{occ}	Valuation function mapping each event $e \in \mathcal{E}$ to the set of all time instants $i \in \mathbb{N}$ at which e occurred (see Definition 2.2).
$\mathcal{T}_{\mathbf{LTL}}(i)$	i-th state of $\mathcal{T}_{\text{LTL}}$ (see Definition 2.2).
$\models_{\mathbf{LTL}}$	LTL Logical Entailment (see Section 2.3)
$\mathbf{CMP}_{\mathbf{LTL}}(\mathcal{T}_{\mathbf{LTL}}, \varphi)$	Compliance of a trace $\mathcal{T}_{\text{LTL}}$ w.r.t. a formula φ based on LTL semantics (see Definition 2.3)
$H(E,T)$	SCIFF notation: an event E has happened at time T (see Section 3.1)
$\mathbf{E}(E,T), \mathbf{EN}(E,T)$	SCIFF notation: an event E is expected to (not) happen at time T, respectively (see Section 3.1)
$\mathcal{S}$	A SCIFF specification, as introduced in Section 3.1
$\mathcal{T}_{\mathbf{SCIFF}}$	A SCIFF execution trace, see Definition 3.1
$\langle \mathcal{S}, \mathcal{T}_{\mathbf{SCIFF}} \rangle$	A SCIFF instance, see Definition 3.2
$\mathbf{CMP}_{\mathbf{SCIFF}}(\mathcal{T}_{\mathbf{SCIFF}}, \mathcal{S})$	Compliance of a trace $\mathcal{T}_{\text{SCIFF}}$ w.r.t. a SCIFF specification $\mathcal{S}$, based on the SCIFF semantics (see Definition 3.8)
`tm`	A transformation between an LTL trace $\mathcal{T}_{\text{LTL}}$ and a SCIFF trace $\mathcal{T}_{\text{SCIFF}}$ (see Definition 4.1).
$\varphi \overset{\mathcal{S}}{\leftrightsquigarrow} \mathcal{S}$	Behavioural equivalence w.r.t. compliance of a LTL formula φ and a SCIFF specification $\mathcal{S}$ (see Definition 4.3)
`icm`	A transformation between a SNF formula to a set of SCIFF integrity constraints (see Definition 5.5)
`sm`	$\mathcal{S}$-mapping, a transformation from a SNF formula to the behaviourally equivalent SCIFF specification $\mathcal{S}$ (see Definition 5.6)
$\mathcal{T}_{\mathbf{SCIFF}}\vert_{\mathcal{V}}$	Trace projection, on the base of the set $\mathcal{V}$, of a SCIFF trace (see Definition 5.9)
$\mathcal{T}_{\mathbf{LTL}}\vert_{\mathcal{V}}$	Trace projection, on the base of the set $\mathcal{V}$, of an LTL trace (see Definition 5.10)