Evaluating Compliance: From LTL to Abductive Logic Programming

Marco Montali[1], Federico Chesani[2], Marco Gavanelli[3], Evelina Lamma[3], and Paola Mello[2]

[1] Free University of Bozen-Bolzano
Piazza Domenicani 3, 39100 – Bolzano (Italy)
montali@inf.unibz.it
[2] University of Bologna
V.le Risorgimento 2, 40136 – Bologna (Italy)
{ federico.chesani | paola.mello }@unibo.it
[3] University of Ferrara
Via Saragat 1, 44122 – Ferrara (Italy)
{ marco.gavanelli | evelina.lamma }@unife.it

Abstract. The *compliance verification* task amounts to establishing if the execution of a system, given in terms of observed 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 this work we review the LTL and the Abductive Logic Programming 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 an automatic translation procedure from LTL to SCIFF which preserves compliance.

Keywords: Linear Temporal Logic, Abductive Logic Programming, Compliance Verification, Business Process Management.

1 Introduction

Linear Temporal Logic (LTL) [12] specifications are traditionally used for expressing the properties that a reactive system should exhibit (or avoid), and are exploited by model checking tools for formal verification (e.g., [15,9]). Recently, LTL has been also used to describe the system under study itself, in fields like Business Process Management (BPM) and Service Oriented Computing (SOC): e.g., the DECLARE system [24] and the ConDec language [23,20] adopt LTL to model business processes, business rules and policies.

In these domains, a relevant task is to assess *compliance*, usually defined as checking if an implementation faithfully meets the requirements of a 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 w.r.t. a *regulatory specification* expressed as an LTL formula. Such approach has been used, for example, in [6] for static compliance verification of BPMN business processes, and in [1] for auditing event logs.

Recently, Logic Programming (LP) based approaches have been applied for specification and verification of normative systems [5,14], web services [25,7] and business processes as well [11,20]. 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. Abductive Logic Programming (ALP, [17]), in particular, integrates abductive reasoning into LP, supporting an hypothesis-making mechanism.

In [2] 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 [4,8], web service interaction [3,21] and BPM [22,20]. SCIFF specifications are given in terms of integrity constraints linking occurring events to expectations about the future course of events, and the declarative semantics has been given in terms of compliance of a given trace with respect to a SCIFF specification.

In this paper we investigate the relation between the LTL-based approach and the SCIFF framework, showing that if we focus on the compliance task, an LTL model can be (formally and correctly) translated into a SCIFF one. Starting from the seminal work in [13] 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 the approaches, we identify a tight equivalence relation, and we prove how such equivalence is indeed maintained when moving from the LTL approach to the SCIFF-based one.

2 Linear Temporal Logic

In this section, we provide a brief introduction to (propositional) Linear-time Temporal Logic (LTL), in particular w.r.t. the notion of *compliance*; the interested reader can refer to [12] for a more general introduction.

LTL formulae are built up from *atomic propositions*, whose truth values change over time. The LTL *time structure* $\mathcal{F}$, also called *frame*, models a single, linear timeline; formally, $\mathcal{F}$ is a totally ordered set $(\mathcal{K}, \prec)$ [12].

Definition 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 $(\mathcal{K}, \prec, v)$ where $v : \mathcal{P} \to 2^{\mathcal{K}}$ 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, each proposition represents a possible event that may occur in an instance of the system. More specifically, a proposition $e \in \mathcal{P}$ is true in a certain state if at that state the event denoted by e occurs. Under this interpretation, LTL models correspond to execution traces.

Definition 2 (LTL execution trace). *Given a set $\mathcal{E}$ of atomic propositions (representing possible events), an* LTL execution trace $\mathcal{T}_{\mathcal{L}}$ *is an LTL model having* $(\mathbb{N}, <)$ *as time structure and $\mathcal{E}$ as the set of atomic propositions. In particular,* $\mathcal{T}_{\mathcal{L}} = (\mathbb{N}, <, v_{occ})$*, where $v_{occ} : \mathcal{E} \rightarrow 2^{\mathbb{N}}$ 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.*

We will use the following abbreviations: $\mathcal{T}_{\mathcal{L}}(i)$ will denote the i-th state of $\mathcal{T}_{\mathcal{L}}$, i.e. the subset $\{e \in \mathcal{E} \mid i \in v_{occ}(e)\}$.

2.1 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). 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 $false \triangleq \neg true$;
– $\Diamond\varphi \triangleq true\mathcal{U}\varphi$, $\Box\varphi \triangleq \neg\Diamond\neg\varphi$ and $\psi\mathcal{W}\varphi \triangleq \psi\mathcal{U}\varphi \vee \Box\psi$.

2.2 Semantics of LTL and Compliance

The semantics of LTL is given w.r.t. an LTL execution trace, and w.r.t. a specific state. We will use $\models_{\mathcal{L}}$ to denote the logical *entailment* in the LTL setting. $\mathcal{M}, i \models_{\mathcal{L}} \varphi$ means that φ is true at time i in model $\mathcal{M}$. $\models_{\mathcal{L}}$ is defined by induction on the structure of the formulae[4]:

$(\mathcal{T}_{\mathcal{L}} \models_{\mathcal{L}} \varphi)$ iff $(\mathcal{T}_{\mathcal{L}}, 0 \models_{\mathcal{L}} \varphi)$;
$(\mathcal{T}_{\mathcal{L}}, i \models_{\mathcal{L}} e)$ iff $e \in \mathcal{T}_{\mathcal{L}}(i)$ (i.e., $i \in v_{occ}(e)$);
$(\mathcal{T}_{\mathcal{L}}, i \not\models_{\mathcal{L}} e)$ iff $e \notin \mathcal{T}_{\mathcal{L}}(i)$;
$(\mathcal{T}_{\mathcal{L}}, i \models_{\mathcal{L}} \neg\varphi)$ iff $(\mathcal{T}_{\mathcal{L}}, i \not\models_{\mathcal{L}} \varphi)$;
$(\mathcal{T}_{\mathcal{L}}, i \models_{\mathcal{L}} \varphi \wedge \psi)$ iff $(\mathcal{T}_{\mathcal{L}}, i \models_{\mathcal{L}} \varphi)$ and $(\mathcal{T}_{\mathcal{L}}, i \models_{\mathcal{L}} \psi)$;
$(\mathcal{T}_{\mathcal{L}}, i \models_{\mathcal{L}} \varphi \vee \psi)$ iff $(\mathcal{T}_{\mathcal{L}}, i \models_{\mathcal{L}} \varphi)$ or $(\mathcal{T}_{\mathcal{L}}, i \models_{\mathcal{L}} \psi)$;
$(\mathcal{T}_{\mathcal{L}}, i \models_{\mathcal{L}} \varphi \Rightarrow \psi)$ iff $(\mathcal{T}_{\mathcal{L}}, i \not\models_{\mathcal{L}} \varphi)$ or $(\mathcal{T}_{\mathcal{L}}, i \models_{\mathcal{L}} \psi)$;
$(\mathcal{T}_{\mathcal{L}}, i \models_{\mathcal{L}} \bigcirc\varphi)$ iff $(\mathcal{T}_{\mathcal{L}}, i+1 \models_{\mathcal{L}} \varphi)$;
$(\mathcal{T}_{\mathcal{L}}, i \models_{\mathcal{L}} \psi\mathcal{U}\varphi)$ iff $\exists k \geq i$ s.t. $(\mathcal{T}_{\mathcal{L}}, k \models_{\mathcal{L}} \varphi)$ and $\forall i \leq j < k$ $(\mathcal{T}_{\mathcal{L}}, j \models_{\mathcal{L}} \psi)$;
$(\mathcal{T}_{\mathcal{L}}, i \models_{\mathcal{L}} \Diamond\varphi)$ iff $\exists j \geq i$ s.t. $(\mathcal{T}_{\mathcal{L}}, j \models_{\mathcal{L}} \varphi)$;
$(\mathcal{T}_{\mathcal{L}}, i \models_{\mathcal{L}} \Box\varphi)$ iff $\forall j \geq i$ $(\mathcal{T}_{\mathcal{L}}, j \models_{\mathcal{L}} \varphi)$;

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

$(\mathcal{T_L}, i \models_{\mathcal{L}} \psi \mathcal{W} \varphi)$ iff either $(\mathcal{T_L}, i \models_{\mathcal{L}} \psi \mathcal{U} \varphi)$ or $(\mathcal{T_L}, i \models_{\mathcal{L}} \Box\psi)$.

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 fulfil them, separating compliant traces from non-compliant ones. In this respect, $\models_{\mathcal{L}}$ plays the role of a *compliance evaluator*.

Definition 3 (LTL Compliance). *An LTL trace $\mathcal{T_L}$ is* compliant *with a LTL formula φ if and only if $\mathcal{T_L}$ entails φ:*

$$\textsc{cmp}_{\text{LTL}}(\mathcal{T_L}, \varphi) \triangleq \mathcal{T_L} \models_{\mathcal{L}} \varphi.$$

When LTL formulae are used to express business constraints/rules of a regulatory model, as for example in the ConDec language [23], 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 w.r.t. a $\mathcal{T_L}$ is verified by means of model checking algorithms.

3 The SCIFF Framework

In the following we will briefly recap the main features of the SCIFF framework. The interested reader can refer to [2] for a detailed and comprehensive presentation. A SCIFF specification $\mathcal{S}$ is an Abductive Logic Program $\langle \mathcal{KB}, \mathcal{A}, \mathcal{IC} \rangle$ [17] where: (i) $\mathcal{KB}$ is a (static) knowledge base (a Logic Program [19]); (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.

Roughly speaking, given a goal $\mathcal{G}$, abductive reasoning looks for a set of literals Δ built from predicates $\mathcal{A}$ such that the goal is entailed by the program $\mathcal{KB} \cup \Delta$, and the set of integrity constraints $\mathcal{IC}$ is entailed too. The set Δ is referred to as an *abductive explanation* (see Definition 6).

Three special predicates are used to model happened events and positive/negative expectations. Happened events are denoted by using the (non abducible) predicate $\mathbf{H}(Ev, T)$, where Ev is a term representing the occurred event, while T explicitly represents the time at which the event occurred. In the remainder of this paper we will assume the time domain relies on natural numbers.

Definition 4 (SCIFF Execution Trace). *A SCIFF* execution trace *$\mathcal{T}$ (or simply a SCIFF trace) is a set of positive ground $\mathbf{H}(E, T)$ atoms.*

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

Definition 5 (SCIFF Instance). *Given a SCIFF specification $\mathcal{S} = \langle \mathcal{KB}, \mathcal{A}, \mathcal{IC} \rangle$ and a trace $\mathcal{T}$, $\langle \mathcal{S}, \mathcal{T} \rangle$ is an* instance *of $\mathcal{S}$.*

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

SCIFF Integrity Constraints (IC) are mainly used to relate happened events with expectations. They are *body* $\rightarrow$ *head* rules, where *body* contains a conjunction of happened events, general abducibles, and defined predicates, while *head* contains a disjunction of conjunctions of expectations, general abducibles, and defined predicates. When the body is matched with events and abducibles, the IC is triggered, and expectations occurring in the head are assumed (abduced).

Definition 6 (Abductive explanation Δ). *Given a SCIFF instance $\langle \mathcal{S}, \mathcal{T} \rangle$, a set $\Delta \subseteq \mathcal{A}$ is an* abductive explanation *for $\langle \mathcal{S}, \mathcal{T} \rangle$ if and only if*

$$\mathrm{Comp}\,(\mathcal{KB} \cup \mathcal{T} \cup \Delta) \cup \mathrm{CET} \cup T_{\mathcal{X}} \models \mathcal{IC}$$

where Comp *is the (two-valued) completion of a theory [18],* CET *stands for Clark Equational Theory [10] and $T_{\mathcal{X}}$ is the CLP constraint theory [16], parametrized by the domain $\mathcal{X}$.*

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

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

together with the usual rules of reflexivity, symmetry and transitivity for equality. Fixing a CLP theory corresponds to instantiate 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 dis-equations. The theory of constraints $T_{\mathbb{N}}$ defines the symbols $+, -, *, /, =, >, <, \geq, \ldots$ with the usual meanings (e.g., $1 < 2 + 2$ is evaluated to *true*).

Remark 1 (Abductive explanations and sub-specifications). If Δ is an abductive explanation for $\langle \mathcal{S}, \mathcal{T} \rangle$, then Δ is an abductive explanation also for $\langle \mathcal{S}' = \langle \mathcal{KB}, \mathcal{A}, \mathcal{IC}' \rangle, \mathcal{T} \rangle$, where $\mathcal{IC}' \subseteq \mathcal{IC}$.

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 framework provides also an *hypotheses-confirmation* mechanism, based on the formal notions of *fulfillment* and *violation.* First of all, expectations must be $\mathbf{E}$-*consistent*: the same event cannot be expected and prohibited at the same time.

Definition 7 (E-consistency). *An abducible set Δ is* **E**-consistent *iff for each event e and for each time t it holds that $\{\mathbf{E}(e,t), \mathbf{EN}(e,t)\} \nsubseteq \Delta$*

The relationship between expectations and happened events is instead captured by the notions of *fulfillment* and *violation*.

Definition 8 ($\mathcal{T}$-Fulfillment). *Given a SCIFF trace $\mathcal{T}$ and an abducible set Δ, Δ is* $\mathcal{T}$-fulfilled *iff for each event e and for each time t: $\mathbf{E}(e,t) \in \Delta \rightarrow \mathbf{H}(e,t) \in \mathcal{T}$ and $\mathbf{EN}(e,t) \in \Delta \rightarrow \mathbf{H}(e,t) \notin \mathcal{T}$*

Definition 9 ($\mathcal{T}$-Violation). *Given a SCIFF trace $\mathcal{T}$ and an abducible set Δ, Δ is* $\mathcal{T}$-violated *iff it exists at least one event e and time t such that: $\mathbf{E}(e,t) \in \Delta \wedge \mathbf{H}(e,t) \notin \mathcal{T}$, or $\mathbf{EN}(e,t) \in \Delta \wedge \mathbf{H}(e,t) \in \mathcal{T}$*

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.

Definition 10 (Compliance in SCIFF). *A trace $\mathcal{T}$ 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 $\langle \mathcal{S}, \mathcal{T} \rangle$;*
2. *Δ is **E**-consistent;*
3. *Δ is $\mathcal{T}$-fulfilled.*

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

4 Relating LTL and SCIFF

LTL and SCIFF rely on different logics, but when capturing regulatory models they both act as compliance evaluators, capturing the same idea of compliance. To capture some similarity w.r.t. compliance, we propose a mapping between LTL and SCIFF. First of all, we need to provide a mapping between an LTL trace $\mathcal{T}_{\mathcal{L}}$ and the corresponding SCIFF trace $\mathcal{T}$ (and vice versa).

Definition 11 (Trace mapping). *Given an LTL trace $\mathcal{T}_{\mathcal{L}} = (\mathbb{N}, <, v_{occ})$ and the set of atomic propositions $\mathcal{E}$, we map any possible pair (e, i) into a corresponding SCIFF event $\mathbf{H}(e,i)$, where $e \in \mathcal{E}$ and $i \in \mathbb{N}$.*

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

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

Example 1. Let us consider an LTL execution trace $\mathcal{T}_{\mathcal{L}} = (\mathbb{N}, <, v_{occ})$, 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 $\mathtt{tm}(\mathcal{T}_{\mathcal{L}}) = \{ \ \mathbf{H}(a,0), \ \mathbf{H}(a,1), \ \mathbf{H}(b,2), \ \mathbf{H}(c,3) \ \}$

The inverse translation, which starts from a SCIFF execution trace and produces a corresponding LTL trace, will be denoted by $\mathtt{tm}^{-1}$.

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 models agree, then they express in some sense "equivalent" prescriptions w.r.t. the trace. Generalizing, if such an agreement is valid for all the possible execution traces, then the two specifications are *behaviorally equivalent.*

Definition 12 (Behavioural equivalence w.r.t. compliance). *A SCIFF specification $\mathcal{S}$ and an LTL formula φ are* behaviorally equivalent w.r.t. compliance *($\varphi \overset{c}{\leftrightsquigarrow} \mathcal{S}$) if and only if* for each *LTL trace $\mathcal{T}_{\mathcal{L}}$ it holds that:*

$$\textsc{cmp}_{\text{LTL}}\left(\mathcal{T}_{\mathcal{L}}, \varphi\right) \Longleftrightarrow \textsc{cmp}_{\text{SCIFF}}\left(\mathit{tm}(\mathcal{T}_{\mathcal{L}}), \mathcal{S}\right).$$

We might notice that Definition 12 does not pose any constraint on the SCIFF specification $\mathcal{S}$: indeed, only the trace $\mathcal{T}_{\mathcal{L}}$ is somehow constrained by the application of the mapping function `tm`.

5 On the Expressiveness of SCIFF

We show now 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) for LTL formulae.

5.1 A Separated Normal Form for LTL Formulae

Fisher and colleagues [13] introduced SNF to express an arbitrary LTL formula by adopting a conjunction of three-basic forms, while preserving satisfiability.

Definition 13 (SNF Formula [13]). *An LTL formula φ is in* SNF *iff φ is a conjunction of formulas of the following forms:*

$$\begin{aligned} \mathbf{start} &\Longrightarrow \bigvee_c l_c && \textit{(an initial LTL-clause)} \\ \Box\Big(\bigwedge_a k_a &\Longrightarrow \bigcirc \bigvee_d l_d\Big) && \textit{(a step LTL-clause)} \\ \Box\Big(\bigwedge_b k_b &\Longrightarrow \Diamond l\Big) && \textit{(a sometime LTL-clause)} \end{aligned}$$

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 14 (LTL to SNF translation [13]). *`snf` is a function which translates an arbitrary LTL formula to a corresponding SNF formula.*

During the transformation, new proposition symbols are introduced to rename complex sub-formulae. Hence, we distinguish between propositions used to represent activities/events, and those used for renaming.

Definition 15 (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 used for renaming during the transformation.*

Example 2. Let us consider LTL "precedence" formula stating that the send_receipt activity can be executed only after having executed the pay activity:

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

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

$$\begin{aligned}\sigma = \texttt{snf}\,[\neg send\_receipt\ \mathcal{W}\ pay] &= \\ = \mathbf{start} \Rightarrow \mathbf{x} \wedge &\begin{cases} \mathbf{start} \Rightarrow (\neg \mathbf{x} \vee \neg send\_receipt \vee pay) \wedge \\ \mathbf{true} \Rightarrow \bigcirc (\neg \mathbf{x} \vee \neg send\_receipt \vee pay) \wedge \\ \mathbf{start} \Rightarrow (\neg \mathbf{x} \vee \mathbf{y} \vee pay) \wedge \\ \mathbf{true} \Rightarrow \bigcirc (\neg \mathbf{x} \vee \mathbf{y} \vee pay) \wedge \\ \mathbf{y} \Rightarrow \bigcirc (\neg send\_receipt \vee pay) \wedge \\ \mathbf{y} \Rightarrow \bigcirc (\mathbf{y} \vee pay) \end{cases}\end{aligned}$$

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 16 ($\mathcal{IC}$-mapping). *An $\mathcal{IC}$-mapping $\texttt{icm}$ is a function which translates an SNF formula to a set of SCIFF integrity constraints, defined as*[5]*:*

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

$$\mathit{icm}\left[\bigwedge_i \varphi_i\right] \triangleq \bigcup_i \mathit{icm}[\varphi_i]$$

$$\mathit{icm}\left[\textbf{start} \implies \bigvee_c l_c\right] \triangleq \mathit{icm}[start, 0] \to \bigvee_c \mathit{icm}[l_c, 0]\,.$$

$$\mathit{icm}\left[\Box\left(\bigwedge_a k_a \implies \bigcirc \bigvee_d l_d\right)\right] \triangleq \bigwedge_a \mathit{icm}[k_a, T] \to \bigvee_d \left(\mathit{icm}[l_d, T_2] \wedge T_2 = T+1\right).$$

$$\mathit{icm}\left[\Box\left(\bigwedge_a k_a \implies \Diamond l\right)\right] \triangleq \bigwedge_a \mathit{icm}[k_a, T] \to \mathit{icm}[l, T_2] \wedge T_2 \geq T.$$

$$\mathit{icm}[start, 0] \triangleq \textbf{occ}(start, 0)$$

$$\mathit{icm}[true, T] \triangleq \textbf{true}(T)$$

$$\mathit{icm}[a, T] \triangleq \textbf{occ}(a, T)$$

$$\mathit{icm}[\neg a, T] \triangleq \textbf{not\_occ}(a, T)$$

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, stating that the proposition *occurs* in that state. Conversely, the absence of the proposition is mapped onto an abducible **not_occ**/2.

Definition 17 ($\mathcal{S}$-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:*

$$\mathit{sm}: \qquad \varphi, \mathcal{V} \longmapsto \langle \emptyset, \{\mathbf{E}/2, \mathbf{EN}/2, \textbf{true}/1, \textbf{occ}/2, \textbf{not\_occ}/2\}, \mathcal{IC})\rangle$$

where

$$\begin{aligned}
\mathcal{IC} = \mathit{icm}(\varphi) \cup \{ & \\
& true \to \textbf{occ}(start, 0). && (S)\\
& true \to \textbf{true}(0). && (T_1)\\
& \textbf{true}(T) \to \textbf{true}(T_2) \wedge T_2 = T+1. && (T_2)\\
& \forall p \in \mathcal{P}(\varphi), p \neq start, \textbf{true}(T) \to \textbf{occ}(p,T) \vee \textbf{not\_occ}(p,T). && (2V)\\
& \textbf{occ}(X,T) \wedge \textbf{not\_occ}(X,T) \to \bot. && (C)\\
& \mathbf{H}(X,T) \wedge X \in \mathcal{V} \to \textbf{occ}(X,T). && (O)\\
& \textbf{occ}(X,T) \wedge X \in \mathcal{V} \to \mathbf{E}(X,T). && (E_1)\\
& \textbf{not\_occ}(X,T) \wedge X \in \mathcal{V} \to \mathbf{EN}(X,T).\ \} && (E_2)
\end{aligned}$$

$\mathcal{S}$-mapping applies $\mathcal{IC}$-mapping and then augments the obtained constraints with further general rules. Such rules capture specific aspects of the LTL semantics:

- (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.
- ($2V$) and (C) are used to model the two-valued semantics of LTL, i.e., that in each state either a proposition is either true or false. 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 (not) occurrence of each proposition in each state with the SCIFF concepts of happened events and expectations.

The next theorem states that `sm` preserves compliance: an arbitrary SNF formula can be translated to a *behaviourally equivalent* SCIFF specification.

Theorem 1 (SCIFF can express SNF formulae). *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. Since LTL and SCIFF share the same semantics for logical symbols AND($\wedge$), OR ($\vee$), and implication($\Rightarrow$ in LTL and $\rightarrow$ in SCIFF), we will focus only on the simplest SNF-forms, consisting of single proposition symbols (instead of conjunctions/disjunctions).

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

If l is a positive literal, say, $l = a$, each compliant LTL execution trace $\mathcal{T}_{\mathcal{L}}$ must satisfy the property that $a \in \mathcal{T}_{\mathcal{L}}(0)$, 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 $\mathcal{T}$ must contain $\mathbf{H}(a, 0)$. 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.

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

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. To disprove $\overset{c}{\rightsquigarrow}$, one must find an execution trace $\mathcal{T}_{\mathcal{L}}$ which is compliant with σ, but whose corresponding trace $\mathcal{T}$ is not compliant with $\mathcal{S} = \mathtt{sm}[\sigma, \mathcal{P}(\sigma)]$. Notice that, by Definition 17 (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 $\mathcal{S}$, $\mathcal{T}$ must contain, for a certain time t the event $\mathbf{H}(k, t)$, while $\mathbf{H}(l, t_2) \notin \mathcal{T}$. By applying the $\mathtt{tm}^{-1}$ function on this trace, one obtains a $\mathcal{T}_{\mathcal{L}}$ which obeys the following properties: (1) $k \in \mathcal{T}_{\mathcal{L}}(t)$, and (2) $l \notin \mathcal{T}_{\mathcal{L}}(t + 1)$. The second property in particular implies that $\mathcal{T}_{\mathcal{L}}$ 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 $\mathcal{T}_{\mathcal{L}}$ trace must obey the following property: $\forall\ t,\ a \in \mathcal{T}_{\mathcal{L}}(t) \vee b \in \mathcal{T}_{\mathcal{L}}(t+1)$. 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 $\mathcal{T}$ must satisfy that $\forall\ t, \mathbf{H}(a, t) \in \mathcal{T} \vee \mathbf{H}(b, t + 1) \in \mathcal{T}$, which is equivalent, under `tm`, to the property on LTL traces.

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

This case of a simple sometime LTL-clause 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

We now demonstrate that also an arbitrary LTL formula can be encoded in SCIFF preserving compliance. The main technical problem is 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 w.r.t. the original LTL formula: positive expectations are imposed also on renaming symbols, which however do not appear in the original LTL formula.

To overcome this issue, the intuitive idea is to restrict the translation `sm` function only to events. The first step is therefore to define, in both settings, a suitable trace projection, which filters an execution trace by maintaining only certain symbols (in particular, the ones which correspond to events).

Definition 18 (SCIFF trace projection). *Given a SCIFF execution trace $\mathcal{T}$ and a set $\mathcal{V}$ of predicate symbols, the* trace projection *of $\mathcal{T}$ on $\mathcal{V}$ ($\mathcal{T}|_{\mathcal{V}}$) is the* *subset of $\mathcal{T}$ containing only events taken from $\mathcal{V}$:*

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

Definition 19 (LTL trace projection). *Given an LTL execution trace $\mathcal{T}_{\mathcal{L}} = (\mathbb{N}, <, v_{occ})$ and a set $\mathcal{V}$ of proposition symbols, the* trace projection *of $\mathcal{T}_{\mathcal{L}}$ on $\mathcal{V}$ ($\mathcal{T}_{\mathcal{L}}|_{\mathcal{V}}$) is the projection of $\mathcal{T}_{\mathcal{L}}$ containing only events taken from $\mathcal{V}$:*

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

Lemma 1 (Commutativity between trace projection and trace mapping). *For each LTL execution trace $\mathcal{T}_{\mathcal{L}}$ and for each set of proposition symbols $\mathcal{V}$*

$$\mathtt{tm}[\mathcal{T}_{\mathcal{L}}|_{\mathcal{V}}] = \mathtt{tm}[\mathcal{T}_{\mathcal{L}}]\,|_{\mathcal{V}}$$

Proof. From the definitions of trace mapping (Def. 11) and of trace projection (Def. 18 and 19).

We now briefly recall one of the main results presented in [13], which proves that SNF preserves satisfiability, i.e., in our setting, that it preserves compliance. Lemma 2 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.

Theorem 2 (SNF preserves satisfiability [13]). *An LTL formula φ is satisfiable iff $\mathtt{snf}(\varphi)$ is satisfiable.*

Lemma 2 (Compliance preservation via extended traces, adapted from [13]). *For each LTL formula φ, it holds that*

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

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

where we remember that (by Definition 15) $\mathcal{E}\left(\mathtt{snf}\left[\varphi\right]\right) = \mathcal{P}(\varphi)$. With such preliminaries, it is possible to prove that each LTL formula is translatable to a SCIFF specification, preserving compliance.

Theorem 3 (SCIFF can express LTL). *Given an arbitrary LTL formula φ and the SCIFF specification $\mathcal{S} = \mathtt{sm}[\mathtt{snf}[\varphi], \mathcal{P}(\varphi)]$, it holds that $\mathcal{S} \overset{c}{\leftrightsquigarrow} \varphi$.*

Proof. Let us denote $\sigma = \mathtt{snf}\left[\varphi\right]$. From Def. 12, 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

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

We will prove firstly one way of the implication ($\Longrightarrow$), and then the opposite direction ($\Longleftarrow$). Both the proofs are organized in the same way: by applying the results obtained in Lemma 1, Lemma 2, and Theorem 1, the problem of proving a formula is reduced to prove 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:

$$\begin{tikzcd}
\forall\ \mathcal{T}_{\mathcal{L}},\ \textsc{cmp}_{\mathrm{LTL}}(\mathcal{T}_{\mathcal{L}}, \varphi) \arrow[r, Rightarrow, "(*)"] \arrow[d, Rightarrow, "\text{Lemma 2}"] & \textsc{cmp}_{\mathrm{SCIFF}}\left(\mathtt{tm}\left[\mathcal{T}_{\mathcal{L}}\right], \mathtt{sm}\left[\sigma, \mathcal{E}(\sigma)\right]\right) \\
\begin{array}{c}\exists \mathcal{T}'_{\mathcal{L}},\ \mathcal{T}_{\mathcal{L}} = \mathcal{T}'_{\mathcal{L}}|_{\mathcal{E}(\sigma)} \\ \wedge\textsc{cmp}_{\mathrm{LTL}}(\mathcal{T}'_{\mathcal{L}}, \sigma)\end{array} \arrow[r, Rightarrow, "\text{Theorem 1}"] & \textsc{cmp}_{\mathrm{SCIFF}}\left(\mathtt{tm}\left[\mathcal{T}'_{\mathcal{L}}\right], \mathtt{sm}\left[\sigma, \mathcal{P}(\sigma)\right]\right) \arrow[u, Rightarrow, "(\dagger)"']
\end{tikzcd}$$

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

$$\textsc{cmp}_{\mathrm{SCIFF}}\left(\mathtt{tm}\left[\mathcal{T}'_{\mathcal{L}}\right], \mathtt{sm}\left[\sigma, \mathcal{P}(\sigma)\right]\right) \Longrightarrow \textsc{cmp}_{\mathrm{SCIFF}}\left(\mathtt{tm}\left[\mathcal{T}'_{\mathcal{L}}|_{\mathcal{E}(\sigma)}\right], \mathtt{sm}\left[\sigma, \mathcal{E}(\sigma)\right]\right) \quad (\dagger)$$

By taking into account abducible sets, Def. 15 and Lemma 1, ($\dagger$) becomes:

$$\textsc{cmp}_{\mathrm{SCIFF}}{}^{\mathit{\Delta}}\left(\mathcal{T}, \mathcal{S}^{\mathcal{E}\mathcal{R}}\right) \Longrightarrow \textsc{cmp}_{\mathrm{SCIFF}}{}^{\mathit{\Delta}'}\left(\mathcal{T}|_{\mathcal{E}(\sigma)}, \mathcal{S}^{\mathcal{E}}\right) \qquad (\ddagger)$$

where $\mathcal{S}^{\mathcal{E}\mathcal{R}} = \mathtt{sm}\left[\sigma, \mathcal{E}(\sigma) \cup \mathcal{R}(\sigma)\right]$, $\mathcal{S}^{\mathcal{E}} = \mathtt{sm}\left[\sigma, \mathcal{E}(\sigma)\right]$ and $\mathcal{T} = \mathtt{tm}\left[\mathcal{T}'_{\mathcal{L}}\right]$. To prove ($\ddagger$), we demonstrate that

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

obeys the three properties required by the Definition 10 of SCIFF compliance:

1. $\mathit{\Delta}'$ is an abductive explanation for $\mathcal{S}^{\mathcal{E}}_{\mathcal{T}|_{\mathcal{E}(\sigma)}}$. The only difference between $\mathcal{S}^{\mathcal{E}}$ and $\mathcal{S}^{\mathcal{E}\mathcal{R}}$ is that, for the first specification, rules (O), (E_1) and (E_2) of Def. 17 do not trigger for events outside $\mathcal{E}(\sigma)$ (in particular, they do not trigger for events inside $\mathcal{R}(\sigma)$). From Remark 1, $\mathit{\Delta}$ 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. $\mathit{\Delta}'$ is **E**-consistent, because $\mathit{\Delta}' \subseteq \mathit{\Delta}$ and $\mathit{\Delta}$ is **E**-consistent.
3. $\mathit{\Delta}'$ is $\mathcal{T}|_{\mathcal{E}(\sigma)}$-fulfilled. Since $\mathcal{T}|_{\mathcal{E}(\sigma)}$ is a projection of $\mathcal{T}$, $\mathit{\Delta}' \subseteq \mathit{\Delta}$ and $\mathit{\Delta}$ is $\mathcal{T}$-fulfilled, no negative expectation in $\mathit{\Delta}'$ can be violated by $\mathcal{T}|_{\mathcal{E}(\sigma)}$. Positive expectations concerning elements in $\mathcal{E}(\sigma)$ are maintained in $\mathit{\Delta}'$, and so are the corresponding happened events after the trace projection. Positive expectations concerning elements in $\mathcal{R}(\sigma)$ are removed from $\mathit{\Delta}$ when obtaining $\mathit{\Delta}'$, and therefore the application of the trace projection, which rules out happened events concerning elements in $\mathcal{R}(\sigma)$, does not affect fulfillment.

($\Longleftarrow$) We move then to prove the other way of the double implication stated in this theorem. Again, let us consider the following schema:

$$\begin{tikzcd}
\textsc{cmp}_{\textrm{LTL}}\left(\mathtt{tm}^{-1}\left[\mathcal{T}\right], \varphi\right) & \forall\, \mathcal{T},\ \textsc{cmp}_{\textrm{SCIFF}}\left(\mathcal{T}, \mathtt{sm}\left[\sigma, \mathcal{E}(\sigma)\right]\right) \arrow[l, Leftarrow, "(**)"'] \arrow[d, Rightarrow, "(\S)"] \\
\textsc{cmp}_{\textrm{LTL}}\left(\mathtt{tm}^{-1}\left[\mathcal{T}'\right], \sigma\right) \arrow[u, Rightarrow, "\text{Lemma 2, then Lemma 1}"'] & \begin{array}{c}\exists\, \mathcal{T}',\ \mathcal{T} = \mathcal{T}'|_{\mathcal{E}(\sigma)} \\ \wedge \textsc{cmp}_{\textrm{SCIFF}}\left(\mathcal{T}', \mathtt{sm}\left[\sigma, \mathcal{P}(\sigma)\right]\right)\end{array} \arrow[l, Leftarrow, "\text{Theorem 1}"']
\end{tikzcd}$$

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

$$\forall\, \mathcal{T},\ {\textsc{cmp}_{\textrm{SCIFF}}}^{\Delta}\left(\mathcal{T}, \mathcal{S}^{\mathcal{E}}\right) \Longrightarrow \exists\, \mathcal{T}',\ \mathcal{T} = \mathcal{T}'|_{\mathcal{E}(\sigma)} \wedge {\textsc{cmp}_{\textrm{SCIFF}}}^{\Delta'}\left(\mathcal{T}', \mathcal{S}^{\mathcal{E}\mathcal{R}}\right) \quad (\S)$$

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

First of all, it is worth noting that $\mathcal{S}^{\mathcal{E}\mathcal{R}}$ 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 $\mathcal{T}' \supseteq \mathcal{T}$, an abductive explanation Δ' can be therefore found for $\mathcal{S}^{\mathcal{E}\mathcal{R}}$ by extending Δ with the new generated expectations: $\Delta' = \Delta \cup \Delta_{\mathcal{R}}^{\mathbf{E}} \cup \Delta_{\mathcal{R}}^{\mathbf{EN}}$, where $\Delta_{\mathcal{R}}^{\mathbf{E}}$ and $\Delta_{\mathcal{R}}^{\mathbf{EN}}$ respectively represent the inserted positive and negative expectations.

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

$$\begin{array}{rl} \forall\ \mathbf{E}(a,t), & \mathbf{E}(a,t) \in \Delta_{\mathcal{R}}^{\mathbf{E}} \Rightarrow \mathbf{occ}(a,t) \in \Delta' \\ \forall\ \mathbf{EN}(a,t), & \mathbf{EN}(a,t) \in \Delta_{\mathcal{R}}^{\mathbf{EN}} \Rightarrow \mathbf{not\_occ}(a,t) \in \Delta' \end{array} \qquad (\S\S)$$

Let us suppose by absurdum that there exist a, t (with $a \in \mathcal{R}(\sigma)$) s.t. $\mathbf{E}(a,t) \in \Delta_{\mathcal{R}}^{\mathbf{E}}$ and $\mathbf{EN}(a,t) \in \Delta_{\mathcal{R}}^{\mathbf{EN}}$. In this case, $(\S\S)$ would state that $\mathbf{occ}(a,t) \in \Delta'$ and $\mathbf{not\_occ}(a,t) \in \Delta'$. This would violate rule (C), making impossible that Δ' is an abductive explanation.

An execution trace $\mathcal{T}^*$ compliant with $\mathcal{S}^{\mathcal{E}\mathcal{R}}$ can be therefore built as follows:

$$\mathcal{T}^* = \mathcal{T} \cup \mathcal{T}^{\mathcal{R}}, \text{ where } \mathbf{H}(a,t) \in \mathcal{T}^{\mathcal{R}} \Leftrightarrow \mathbf{E}(a,t) \in \Delta_{\mathcal{R}}^{\mathbf{E}}$$

Under this choice:

1. Δ' is left untouched by $\mathcal{T}^*$. Indeed, the only impact of $\mathcal{T}^{\mathcal{R}}$ on the ICs of $\mathcal{S}^{\mathcal{E}\mathcal{R}}$ is to trigger rule (O), generating corresponding **occ** abducibles. However, from $(\S\S)$ we know that all these abducibles are already contained in Δ'.
2. Δ' is $\mathcal{T}^*$-fulfilled by construction.
3. $\mathcal{T}^*|_{\mathcal{E}(\sigma)} = \mathcal{T}$, because all the happened events contained in $\mathcal{T}^{\mathcal{R}}$ involve symbols belonging to $\mathcal{R}(\sigma)$, and are therefore ruled out by applying the projection.

6 Discussion and Conclusion

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 we provide an automatic translation between LTL-based and SCIFF-based specifications. We prove that such translation preserves the notion of compliance, w.r.t. the defined equivalence.

LTL-based techniques for verification have a number of strengths and weaknesses, as well as the SCIFF framework that inherits advantages and limits of LP 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 Computational Logic, where the best of both worlds (LTL and SCIFF) can be coherently exploited. E.g., LP-based approaches support the use of variables and constraints over them, allowing to model systems where also data and data relations/constraints are taken into account.

A limit of the presented approach stems from the semi-decidability issues of (refutation-based) logic programming. SCIFF inherits such characteristics: as a consequence, not any possible LTL specification could be directly reasoned about in SCIFF. From an operational viewpoint the problem can be avoided by restricting to a significant fragment of LTL, and provide ad-hoc translations for which termination is guaranteed. Alternatively, it is possible to notice that a number of applications inherently require finite traces, like e.g. business processes [23], that are developed to reach a business goal (such as delivery a product) in a finite number of steps. Automatic translation of any LTL formula within a finite trace semantics into a SCIFF corresponding model is matter of ongoing work.

References

1. 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. LNCS, vol. 3760, pp. 130–147. Springer (2005)
2. 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 9(4), 29:1–29:43 (Aug 2008)
3. Alberti, M., Gavanelli, M., Lamma, E., Chesani, F., Mello, P., Montali, M.: An abductive framework for a-priori verification of web services. In: Bossi, A., Maher, M.J. (eds.) Procs. of PPDP 2006, July 10-12, 2006, Venice, Italy. pp. 39–50. ACM, New York, USA (2006)
4. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P., Sartor, G.: Mapping deontic operators to abductive expectations. Computational & Mathematical Organization Theory 12(2-3), 205–225 (2006)
5. Artikis, A., Sergot, M., Pitt, J.: Specifying Norm-Governed Computational Societies. ACM Transactions on Computational Logic 10(1), 1–42 (2009)
6. Awad, A., Decker, G., Weske, M.: Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In: Dumas, M., Reichert, M., Shan, M.C. (eds.) 6th Intl. Conf. BPM 2008. LNCS, vol. 5240, pp. 326–341. Springer (2008)

7. 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 70(1), 53–73 (2007)
8. 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). pp. 91–96 (2009)
9. Cimatti, A., Clarke, E.M., 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, K.G. (eds.) CAV. Lecture Notes in Computer Science, vol. 2404, pp. 359–364. Springer (2002)
10. Clark, K.L.: Negation as Failure. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 293–322. Plenum Press (1978)
11. 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. pp. 170–181. Rimini, Italy (2010)
12. Emerson, E.A.: Temporal and Modal Logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics. Elsevier and MIT Press (1990)
13. Fisher, M., Dixon, C., Peim, M.: Clausal Temporal Resolution. ACM Transactions on Computational Logic 2(1), 12–56 (2001)
14. Fornara, N., Colombetti, M.: Specifying artificial institutions in the event calculus. In: Dignum, V. (ed.) Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models. pp. 335–366. IGI Global (2009)
15. Holzmann, G.: The model checker spin. Software Engineering, IEEE Transactions on 23(5), 279–295 (1997)
16. Jaffar, J., Maher, M.J., Marriott, K., Stuckey, P.J.: The semantics of constraint logic programs. J. Log. Program. 37(1-3), 1–46 (1998)
17. Kakas, A.C., Kowalski, R.A., Toni, F.: Abductive Logic Programming. Journal of Logic and Computation 2(6), 719–770 (1993)
18. Kunen, K.: Negation in logic programming. J. Log. Program. 4(4), 289–308 (1987)
19. Lloyd, J.W.: Foundations of Logic Programming. Springer, 2nd edn. (1987)
20. Montali, M.: Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach, LNBIP, vol. 56. Springer (2010)
21. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative Specification and Verification of Service Choreographies. ACM Transactions on the Web 4(1) (2010)
22. 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 102(3-4), 325–361 (2010)
23. Pesic, M., van der Aalst, W.M.P.: A Declarative Approach for Flexible Business Processes Management. In: Eder, J., Dustdar, S. (eds.) Procs. of the BPM 2006 Workshops. LNCS, vol. 4103, pp. 169–180. Springer (2006)
24. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: Full Support for Loosely-Structured Processes. In: Procs. IEEE EDOC 2007. pp. 287–300. IEEE Computer Society (2007)
25. Roman, D., Kifer, M.: Semantic Web Service Choreography: Contracting and Enactment. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T.W., Thirunarayan, K. (eds.) Procs. ISWC 2008. LNCS, vol. 5318, pp. 550–566. Springer (2008)