Diagnosis of Event Sequences with LFIT

Tony Ribeiro[1], Maxime Folschette[2], Morgan Magnin[3,4], Kotaro Okazaki[5], Lo Kuo-Yen[5], and Katsumi Inoue[4]

[1] Independent Researcher
[2] Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
[3] Univ. Nantes, CNRS, Centrale Nantes, UMR 6004 LS2N, F-44000 Nantes, France
[4] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
[5] SONAR Inc., 8-16-6, Ginza, Chuo-Ku, Tokyo 104-0061, Japan

Abstract. Diagnosis of the traces of executions of discrete event systems is of interest to understand dynamical behaviors of a wide range of real world problems like real-time systems or biological networks. In this paper, we propose to address this challenge by extending Learning From Interpretation Transition (LFIT), an Inductive Logic Programming framework that automatically constructs a model of the dynamics of a system from the observation of its state transitions. As a way to tackle diagnosis, we extend the theory of LFIT to model event sequences and their temporal properties. It allows to learn logic rules that exploit those properties to explain sequences of interest. We show how it can be done in practice through a case study.

Keywords: dynamic systems · logical modeling · explainable artificial intelligence

1 Introduction

Discrete event systems have been formalized as a wide range of paradigms, e.g., Petri nets [6], to model dynamical behaviors. In this paper, we propose to focus on learning dynamical properties from the trace of executions of such system, i.e., sequences of events. Such a setting can be related to fault diagnosis, which has been the subject of much interest [5]. It consists of identifying underlying phenomena that result in the failure of a system. It takes as input a model and a set of observations of the system under the form of event sequences. In our case, we only consider the event sequences as input and propose a method independent of the model paradigm.

Since its first establishment in the 80s and 90s, Inductive Logic Programming (ILP) has been identified as a promising approach to tackle such a diagnosis problem [4] and several works followed [3,10]. Learning From Interpretation Transition (LFIT) [2] is an ILP framework that automatically builds a model of

ILP 2022, 31st International Conference on Inductive Logic Programming, Cumberland Lodge, Windsor, UK

the dynamics of a system from the observation of its state transitions. Our goal here is to extend LFIT to exploit temporal properties to explain event sequences of interest. Figure 1 illustrates the general LFIT learning process. Given some raw data, like time-series of gene expression, a discretization of those data in the form of state transitions is assumed. From those state transitions, according to the semantics of the system dynamics, several inference algorithms modeling the system as a logic program have been proposed.

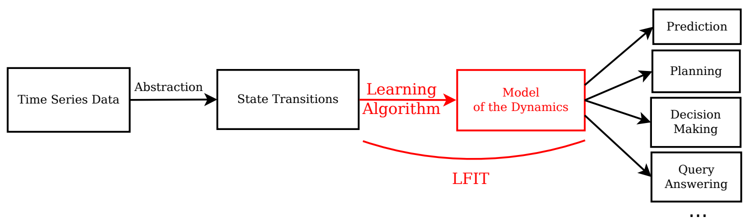

Fig. 1: Assuming a discretization of time series data of a system as state transitions, we propose a method to automatically model the system dynamics.

In [8], we extended this framework to learn system dynamics independently of its update semantics. For this purpose, we proposed a modeling of discrete memory-less multi-valued systems as logic programs in which each rule represents the possibility for a variable to take some value in the next state. This modeling permits to characterize optimal programs independently of the update semantics, allowing to model the dynamics of a wide range of discrete systems. To learn such semantic-free optimal programs, we proposed **GULA**: the General Usage LFIT Algorithm that now serves as the core block to several methods of the framework. In this paper, we show how to use **GULA** in order to learn logic rules that combine temporal patterns to explain event sequences of interest. We use a case study to show some of the difficulties and interests of the method.

2 Dynamical Multi-Valued Logic Program

In this section, the concepts necessary to understand the modeling we propose in this paper are formalized. Let $\mathcal{V} = \{\mathrm{v}_1, \cdots, \mathrm{v}_n\}$ be a finite set of $n \in \mathbb{N}$ variables, $\mathcal{V}al$ the set in which variables take their values and $\mathsf{dom} : \mathcal{V} \to \wp(\mathcal{V}al)$ a function associating a domain to each variable, with $\wp$ the power set. The atoms of *multi-valued logic* ($\mathcal{M}$VL) are of the form v^{val} where $\mathrm{v} \in \mathcal{V}$ and $val \in \mathsf{dom}(\mathrm{v})$. The set of such atoms is denoted by $\mathcal{A} = \{\mathrm{v}^{val} \in \mathcal{V} \times \mathcal{V}al \mid val \in \mathsf{dom}(\mathrm{v})\}$. Let $\mathcal{F}$ and $\mathcal{T}$ be a partition of $\mathcal{V}$, that is: $\mathcal{V} = \mathcal{F} \cup \mathcal{T}$ and $\mathcal{F} \cap \mathcal{T} = \emptyset$. $\mathcal{F}$ is called the set of *feature* variables, which values represent the state of the system at the previous time step $(t-1)$, and $\mathcal{T}$ is called the set of *target* variables, which values represent the state of the system at the current time step (t). A $\mathcal{M}$VL *rule* R is defined by:

$$R \quad = \quad \mathrm{v}_0^{val_0} \leftarrow \mathrm{v}_1^{val_1} \wedge \cdots \wedge \mathrm{v}_m^{val_m}$$

where $m \in \mathbb{N}$, and $\forall i \in [\![0; m]\!], \mathrm{v}_i^{val_i} \in \mathcal{A}$; furthermore, every variable is mentioned at most once in the right-hand part: $\forall j, k \in [\![1; m]\!], j \neq k \Rightarrow \mathrm{v}_j \neq \mathrm{v}_k$. The rule R has the following meaning: the variable v_0 can take the value val_0 in the next dynamical step if for each $i \in [\![1; m]\!]$, variable v_i has value val_i in the current dynamical step. The atom on the left side of the arrow is called the *head* of R, denoted $\mathrm{head}(R) := \mathrm{v}_0^{val_0}$, and is made of a target variable: $\mathrm{v}_0 \in \mathcal{T}$. The notation $\mathrm{var}(\mathrm{head}(R)) := \mathrm{v}_0$ denotes the variable that occurs in $\mathrm{head}(R)$. The conjunction on the right-hand side of the arrow is called the *body* of R, written $\mathrm{body}(R)$, and all variables in the body are feature variables: $\forall i \in [\![1; m]\!], \mathrm{v}_i \in \mathcal{F}$. In the following, the body of a rule is assimilated to the set $\{\mathrm{v}_1^{val_1}, \cdots, \mathrm{v}_m^{val_m}\}$; we thus use set operations such as $\in$ and $\cap$ on it, and we denote $\emptyset$ an empty body. A *dynamical multi-valued logic program* ($\mathcal{DM}$VLP) is a set of $\mathcal{M}$VL rules.

Definition 1 (Rule Domination). *Let R_1, R_2 be $\mathcal{M}$VL rules. R_1* dominates *R_2, written $R_1 \geq R_2$ if $\mathrm{head}(R_1) = \mathrm{head}(R_2)$ and $\mathrm{body}(R_1) \subseteq \mathrm{body}(R_2)$.*

The dynamical system we want to learn the rules of, is represented by a succession of *states* as formally given by Definition 2. We also define the "compatibility" of a rule with a state in Definition 3, and with a transition in Definition 4.

Definition 2 (Discrete state). *A* discrete state *s on a set of variables $\mathcal{X}$ of a $\mathcal{DM}$VLP is a function from $\mathcal{X}$ to $(\mathsf{dom}(\mathrm{v}))_{\mathrm{v} \in \mathcal{X}}$. It can be equivalently represented by the set of atoms $\{\mathrm{v}^{s(\mathrm{v})} \mid \mathrm{v} \in \mathcal{X}\}$ and thus we can use classical set operations on it. We write $\mathcal{S}^{\mathcal{X}}$ to denote the set of all discrete states of $\mathcal{X}$.*

Often, $\mathcal{X} \in \{\mathcal{F}, \mathcal{T}\}$. In particular, a couple of states $(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ is called a *transition*.

Definition 3 (Rule-state matching). *Let $s \in \mathcal{S}^{\mathcal{F}}$. The $\mathcal{M}$VL rule R* matches *s, written $R \sqcap s$, if $\mathrm{body}(R) \subseteq s$.*

The final program we want to learn should both: (1) match the observations in a complete (all transitions are learned) and correct (no spurious transition) way; (2) represent only minimal necessary interactions (no overly-complex rules). The following definitions formalize these desired properties.

Definition 4 (Rule and program realization). *Let R be a $\mathcal{M}$VL rule and $(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. The rule R* realizes *the transition (s, s') if $R \sqcap s \wedge \mathrm{head}(R) \in s'$. A $\mathcal{DM}$VLP P* realizes *(s, s') if $\forall \mathrm{v} \in \mathcal{T}, \exists R \in P, \mathrm{var}(\mathrm{head}(R)) = \mathrm{v} \wedge R$ realizes (s, s'). P* realizes *a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ if $\forall (s, s') \in T, P$ realizes (s, s').*

Definition 5 (Conflict and Consistency). *A $\mathcal{M}$VL rule R* conflicts *with a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ when $\exists (s, s') \in T, \big(R \sqcap s \wedge \forall (s, s'') \in T, \mathrm{head}(R) \notin s''\big)$. Otherwise, R is said to be* consistent *with T. A $\mathcal{DM}$VLP P is* consistent *with a set of transitions T if P does not contain any rule R conflicting with T.*

Definition 6 (Suitable and optimal program). *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. A $\mathcal{DM}$VLP P* is suitable for *T if: P is consistent with T, P realizes T, and for any possible $\mathcal{M}$VL rule R consistent with T, there exists $R' \in P$ s.t. $R' \geq R$. If in addition, for all $R \in P$, all the $\mathcal{M}$VL rules R' belonging to $\mathcal{DM}$VLP suitable for T are such that $R' \geq R$ implies $R \geq R'$, then P is called* optimal *and denoted $P_{\mathcal{O}}(T)$.*

In [8], we proposed the General Usage LFIT Algorithm (**GULA**) that guarantees to learn the optimal program of a set of transitions: let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, $\textbf{GULA}(\mathcal{A}, T, \mathcal{F}, \mathcal{T}) = P_{\mathcal{O}}(T)$ (Theorem 5 of [8]).

The present work builds upon the definitions presented above. The aim is to use GULA to learn about the possible influence of additional properties along the original observations. If those properties respect the following proposition, they can appear in rules learned by **GULA**, encoded as regular $\mathcal{M}$VL atoms.

Proposition 1 (Properties encoding).

- *Let* $\mathcal{V} := \mathcal{V}_{\mathcal{F}} \cup \mathcal{V}_{\mathcal{T}}$ *a set of* $\mathcal{M}$VL *feature and target variables;*
- *Let* $\mathcal{A} := \mathcal{A}_{\mathcal{F}} \cup \mathcal{A}_{\mathcal{T}}$ *be the corresponding feature and target atoms;*
- *Let* $\mathcal{S}^{\mathcal{F}} \subseteq \mathcal{A}_{\mathcal{F}}$ *be feature states (one atom of* $\mathcal{A}_{\mathcal{F}}$ *per variable of* $\mathcal{V}_{\mathcal{F}}$*);*
- *Let* $\mathcal{V}_P$ *be a set of variables,* $\mathcal{V}_P \cap \mathcal{V} = \emptyset$*, and* $\mathcal{A}_P$ *the corresponding atoms;*
- *Let* $P : \mathcal{S}^{\mathcal{F}} \to \mathcal{S}^{\mathcal{V}_P}$ *a function computing a* property *on feature states;*
- *Let* $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ *be a set of transitions;*
- *Let* $T' := \{(s \cup P(s), s') \mid (s, s') \in T\}$ *be the encoding of property* P *on* T*;*
- *Then,* $\textbf{GULA}(\mathcal{A} \cup \mathcal{A}_P, T', \mathcal{F} \cup \mathcal{V}_P, \mathcal{T}) = P_{\mathcal{O}}(T')$ *and the rules of* $P_{\mathcal{O}}(T')$ *contain atoms of property* P *only if it is necessary to realize a target.*

Proof sketch. By construction from Theorem 1 of [8] and from Definition 6. □

Proposition 1 allows to encode additional properties of the observation as regular **GULA** input. For a given target atom, the atoms corresponding to a property will appear in the rules of the optimal logic program only if the property is a necessary condition to obtain this target atom. One use of such encoding is to obtain more understandable rules as shown in the following sections.

3 Diagnosis of Labelled Event Sequences

Event sequences have the advantages to be considered as raw output data for many dynamical systems while being able to represent the dynamics of a large set of discrete models (Petri nets, logic programs, ...). As such, it is easy to use them to assert the set of desirable (or undesirable) sequences. In this section, we propose a modeling of event sequences and their temporal properties into the LFIT framework. It allows to use **GULA** to learn logic rules that exploit those properties to explain sequences of interest.

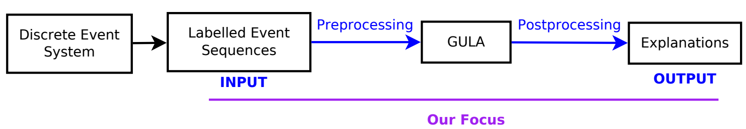

Fig. 2: This paper focuses on the modeling and encoding of labelled event sequence for **GULA** to learn explanation rules exploiting temporal properties.

3.1 Modeling Labelled Event Sequences

Definition 7 (Sequence). *A* sequence *s is a tuple $s = (s_i)_{i\in[0,|s|-1]}$. In the rest of the paper, we note s_i the i^{th} element of s.*

An event sequence classification problem (ESCP) is a triple $(E, Seq_{pos}, Seq_{neg})$:
- $E = \{e_0, \ldots, e_m\}$ is a set of elements called *events*;
- $Seq \subseteq E^n$ is a set of sequences of events of size $n \in \mathbb{N}$;
- $Seq_{pos} \subseteq Seq$ is the set of positive examples;
- $Seq_{neg} \subseteq Seq$ is the set of negative examples;
- $Seq_{pos} \cap Seq_{neg} = \emptyset$.

Such classification problem can be encoded into $\mathcal{M}$VL, allowing **GULA** to learn a classifier in the form of a $\mathcal{DM}$VLP. The algorithm takes as input a set of atoms $\mathcal{A}$, a set of transitions T, a set of feature variables $\mathcal{F}$ and a set of target variables $\mathcal{T}$. An ESCP can be encoded as follows.

Proposition 2 ($\mathcal{M}$VL encoding of ESCP). *Let $(E, Seq_{pos}, Seq_{neg})$ be an ESCP. The encoding of this ESCP is done as follows:*
- $\mathcal{A} := \{\mathsf{ev}_i^e \mid 0 \le i < n, e \in E\} \cup \{\mathsf{label}^{pos}, \mathsf{label}^{neg}\}$ *the set of $\mathcal{M}$VL atoms;*
- $\mathcal{F} := \{\mathsf{ev}_i \mid 0 \le i < n\}$ *the set of feature variables;*
- $\mathcal{T} := \{\mathsf{label}\}$ *the set of target variables;*
- $f : Seq \to \mathcal{S}^{\mathcal{F}}$ *with* $s \overset{f}{\mapsto} \{\mathsf{ev}_i^e \in \mathcal{A} \mid i \in [0, |s|] \wedge s_i = e\}$ *to encode positions;*
- $T := \{(f(s), \{\mathsf{label}^{pos}\}) \mid s \in Seq_{pos}\} \cup \{(f(s), \{\mathsf{label}^{neg}\}) \mid s \in Seq_{neg}\}$ *the set of transitions.*

Example 1. Let us consider an ESCP with 3 events and sequences of size 4. Consider the following ESCP $(E, Seq_{pos}, Seq_{neg})$ where not all sequences are detailed:
- $E = \{e_0, e_1, e_2\}$
- $Seq_{pos} = \{(e_1, e_1, e_0, e_2), (e_1, e_0, e_1, e_2), (e_1, e_0, e_0, e_2), (e_1, e_0, e_2, e_1), \ldots\}$
- $Seq_{neg} = \{(e_1, e_1, e_1, e_1), (e_1, e_1, e_1, e_0), (e_1, e_1, e_1, e_2), (e_1, e_1, e_0, e_1), \ldots\}$

Now consider the corresponding $\mathcal{M}$VL encoding: $(\mathcal{A}, T, \mathcal{F}, \mathcal{T})$:
- $A = \{\mathsf{ev}_0^{e_0}, \mathsf{ev}_0^{e_1}, \mathsf{ev}_0^{e_2}, \mathsf{ev}_1^{e_0}, \mathsf{ev}_1^{e_1}, \ldots\} \cup \{\mathsf{label}^{pos}, \mathsf{label}^{neg}\}$
- $\mathcal{F} = \{\mathsf{ev}_0, \mathsf{ev}_1, \mathsf{ev}_2, \mathsf{ev}_3\}$
- $\mathcal{T} = \{\mathsf{label}\}$
- $T = \{(\{\mathsf{ev}_0^{e_1}, \mathsf{ev}_1^{e_1}, \mathsf{ev}_2^{e_0}, \mathsf{ev}_3^{e_2}\}, \{\mathsf{label}^{pos}\}), (\{\mathsf{ev}_0^{e_1}, \mathsf{ev}_1^{e_0}, \mathsf{ev}_2^{e_1}, \mathsf{ev}_3^{e_2}\}, \{\mathsf{label}^{pos}\}), \ldots$
  $(\{\mathsf{ev}_0^{e_1}, \mathsf{ev}_1^{e_1}, \mathsf{ev}_2^{e_1}, \mathsf{ev}_3^{e_1}\}, \{\mathsf{label}^{neg}\}), (\{\mathsf{ev}_0^{e_1}, \mathsf{ev}_1^{e_1}, \mathsf{ev}_2^{e_1}, \mathsf{ev}_3^{e_0}\}, \{\mathsf{label}^{neg}\}), \ldots\}$

Using the encoding of Proposition 2, the call to $\mathbf{GULA}(\mathcal{A}, T, \mathcal{F}, \mathcal{T})$ will output a set of rules P such that using rule matching (Definition 3) we obtain a correct classifier, as stated by Theorem 1. Indeed, all rules of P that match a positive or negative observation has the correct label as head and there is always at least one rule that matches each observation.

Theorem 1. *Let $(E, Seq_{pos}, Seq_{neg})$ be an ESCP. Let $\mathcal{A}, T, \mathcal{F}, \mathcal{T}, f$ be as in Proposition 2. The following holds:*

$$\forall l \in \{pos, neg\}, \forall s \in Seq_l, \{\text{head}(R) \mid R \in \mathbf{GULA}(\mathcal{A}, T, \mathcal{F}, \mathcal{T}), R \sqcap f(s)\} = \{\mathsf{label}^l\}$$

Proof sketch. By construction from Theorem 1 of [8], Definition 6 and Proposition 2. □

To ease rule readability in the following examples, atom a^i is written $a(i)$.

Example 2. Let us consider the set of events $E := \{e_0, e_1, e_2\}$ and sequences of size 4. The following set of positive examples $Seq_{pos} = \{ (e_1, e_1, e_0, e_2), (e_1, e_0, e_1, e_2), (e_1, e_0, e_0, e_2), (e_1, e_0, e_2, e_1), (e_1, e_0, e_2, e_2), (e_0, e_1, e_1, e_2), (e_0, e_1, e_0, e_2), (e_0, e_1, e_2, e_1), (e_0, e_1, e_2, e_2), (e_0, e_0, e_1, e_2), (e_0, e_0, e_0, e_2), (e_0, e_0, e_2, e_1), (e_0, e_0, e_2, e_2), (e_0, e_2, e_1, e_1), (e_0, e_2, e_1, e_2), (e_0, e_2, e_2, e_1), (e_0, e_2, e_2, e_2) \}$.

All other possible sequences are negative examples: $Seq_{neg} = Seq \setminus Seq_{pos}$, thus: $Seq_{neg} = \{ (e_1, e_1, e_1, e_1), (e_1, e_1, e_1, e_0), (e_1, e_1, e_1, e_2), (e_1, e_1, e_0, e_1), (e_1, e_1, e_0, e_0), (e_1, e_1, e_2, e_1), (e_1, e_1, e_2, e_0), \ldots, (e_2, e_2, e_2, e_0), (e_2, e_2, e_2, e_2) \}$.

Using the encoding of Proposition 2 we obtain the following :

$\mathbf{GULA}(\mathcal{A}, T, \mathcal{F}, \mathcal{T}) =$

$\mathsf{label}(pos) \leftarrow \mathsf{ev}_0(e_0), \mathsf{ev}_1(e_1), \mathsf{ev}_3(e_2).$
$\mathsf{label}(pos) \leftarrow \mathsf{ev}_0(e_0), \mathsf{ev}_2(e_1), \mathsf{ev}_3(e_2).$
$\mathsf{label}(pos) \leftarrow \mathsf{ev}_0(e_0), \mathsf{ev}_2(e_2), \mathsf{ev}_3(e_1).$
$\mathsf{label}(pos) \leftarrow \mathsf{ev}_0(e_0), \mathsf{ev}_2(e_2), \mathsf{ev}_3(e_2).$
$\mathsf{label}(pos) \leftarrow \mathsf{ev}_0(e_0), \mathsf{ev}_1(e_2), \mathsf{ev}_2(e_1), \mathsf{ev}_3(e_1).$
$\mathsf{label}(pos) \leftarrow \mathsf{ev}_0(e_0), \mathsf{ev}_1(e_0), \mathsf{ev}_3(e_2).$
$\mathsf{label}(pos) \leftarrow \mathsf{ev}_0(e_1), \mathsf{ev}_1(e_0), \mathsf{ev}_3(e_2).$
$\mathsf{label}(pos) \leftarrow \mathsf{ev}_0(e_1), \mathsf{ev}_1(e_0), \mathsf{ev}_2(e_2), \mathsf{ev}_3(e_1).$
$\mathsf{label}(pos) \leftarrow \mathsf{ev}_0(e_1), \mathsf{ev}_1(e_1), \mathsf{ev}_2(e_0), \mathsf{ev}_3(e_2).$

$\mathsf{label}(neg) \leftarrow \mathsf{ev}_0(e_2).$
$\mathsf{label}(neg) \leftarrow \mathsf{ev}_3(e_0).$
$\mathsf{label}(neg) \leftarrow \mathsf{ev}_2(e_0), \mathsf{ev}_3(e_1).$
$\mathsf{label}(neg) \leftarrow \mathsf{ev}_1(e_2), \mathsf{ev}_2(e_0).$
$\mathsf{label}(neg) \leftarrow \mathsf{ev}_1(e_0), \mathsf{ev}_2(e_1), \mathsf{ev}_3(e_1).$
$\mathsf{label}(neg) \leftarrow \mathsf{ev}_1(e_1), \mathsf{ev}_2(e_1), \mathsf{ev}_3(e_1).$
$\mathsf{label}(neg) \leftarrow \mathsf{ev}_0(e_1), \mathsf{ev}_1(e_2).$
$\mathsf{label}(neg) \leftarrow \mathsf{ev}_0(e_1), \mathsf{ev}_2(e_1), \mathsf{ev}_3(e_1).$
$\mathsf{label}(neg) \leftarrow \mathsf{ev}_0(e_1), \mathsf{ev}_1(e_1), \mathsf{ev}_2(e_1).$
$\mathsf{label}(neg) \leftarrow \mathsf{ev}_0(e_1), \mathsf{ev}_1(e_1), \mathsf{ev}_2(e_2).$
$\mathsf{label}(neg) \leftarrow \mathsf{ev}_0(e_1), \mathsf{ev}_1(e_1), \mathsf{ev}_3(e_1).$

This $\mathcal{DM}$VLP correctly classifies each sequence of Seq_{pos} and Seq_{neg} (see Theorem 1) but position atoms (ev) are not enough to explain simply the whole dynamics.

In Example 2, the encoding of Proposition 2 is arguably not enough for the rules to explicitly explain the real influence of the system. The positive rules (whose head is $\mathsf{label}(pos)$) are very specific and it is not easy to make sense from them individually. But we can see at least that all of them contain both e_0 and e_2, thus their relationship must be of importance. Some negative rules are of interest too: the two first ones tell us that e_2 cannot start the sequence ($\mathsf{label}(neg) \leftarrow \mathsf{ev}_0(e_2).$) and e_0 cannot finish it ($\mathsf{label}(neg) \leftarrow \mathsf{ev}_3(e_0).$), thus their ordering is also of importance.

With this simple encoding, the learned rules are mere consequences of the real property behind this example, but none of them fully represents the property itself. In order to have more meaningful rules, we could encode some properties of interest as new variables and atoms by following Proposition 1. The idea is to propose an encoding of some simple general temporal property which can be combined to capture and explain the hidden property of the observed system.

3.2 Encoding Elementary LTL Operators

Linear Temporal Logic (LTL) [7] is a modal temporal logic used to characterize the occurrence of properties in a unique linear dynamical path, like the event sequences studied in this paper. It is mainly composed of the following operators:

- F(ϕ): ϕ eventually has to hold (Finally);
- G(ϕ): ϕ has to hold on the entire subsequent path (Globally);
- U(ψ, ϕ): ψ has to hold at least until ϕ becomes true, which must hold at the current or a future position (Until).

These operators over a sequence s can be encoded into a feature state following Proposition 1 and the interpretation given below:

- $Finally(s, e) \equiv e \in s$
- $Globally(s, e) \equiv e' \in s \implies e' = e$
- $Until(s, e_1, e_2) \equiv \exists i \in [1, |s|], s_i = e_2 \wedge \forall j \in [1, i-1], s_j = e_1$

Example 3. Following Proposition 1, we can encode those LTL properties as additional $\mathcal{M}$VL variables and atoms: $\mathcal{V}_P = \{\ F\_e_0,\ F\_e_1,\ F\_e_2,\ G\_e_0,\ G\_e_1,\ G\_e_2,\ U\_e_0\_e_1, U\_e_0\_e_2, U\_e_1\_e_0, U\_e_1\_e_2, U\_e_2\_e_0, U\_e_2\_e_1,\ \}$, with $\forall \mathrm{v} \in \mathcal{V}_P, \mathsf{dom}(\mathrm{v}) = \{true, false\}$, where F_e_i encodes $Finally(s, e_i)$, G_e_i encodes $Globally(s, e_i)$, $U\_e_i\_e_j$ encodes $Until(s, e_i, e_j)$.
Using this encoding on the transitions T of example 2 we obtain:

$$T' = \{(\{\mathsf{ev}_0^{e_1}, \mathsf{ev}_1^{e_1}, \mathsf{ev}_2^{e_0}, \mathsf{ev}_3^{e_2}, F\_e_0^{true}, F\_e_1^{true}, F\_e_2^{true}, G\_e_0^{false}, \ldots\}, \{\mathsf{label}^{pos}\}),$$
$$(\{\mathsf{ev}_0^{e_1}, \mathsf{ev}_1^{e_0}, \mathsf{ev}_2^{e_1}, \mathsf{ev}_3^{e_2}, F\_e_0^{true}, F\_e_1^{true}, F\_e_2^{true}, G\_e_0^{false}, \ldots\}, \{\mathsf{label}^{pos}\}),$$
$$\ldots$$
$$(\{\mathsf{ev}_0^{e_2}, \mathsf{ev}_1^{e_2}, \mathsf{ev}_2^{e_2}, \mathsf{ev}_3^{e_0}, F\_e_0^{true}, F\_e_1^{false}, F\_e_2^{true}, G\_e_0^{false}, \ldots\}, \{\mathsf{label}^{neg}\}),$$
$$(\{\mathsf{ev}_0^{e_2}, \mathsf{ev}_1^{e_2}, \mathsf{ev}_2^{e_2}, \mathsf{ev}_3^{e_2}, F\_e_0^{false}, F\_e_1^{false}, F\_e_2^{true}, G\_e_0^{false}, \ldots\}, \{\mathsf{label}^{neg}\})\}$$

We can now use **GULA** to learn rules that exploit those encoded properties.
$\mathbf{GULA}(\mathcal{A} \cup \mathcal{A}_P, T', \mathcal{F} \cup \mathcal{V}_P, \mathcal{T})$:

$\mathsf{label}(pos) \leftarrow ev_3(e_2), F\_e_1(true), U\_e_1\_e_2(false).$ $\mathsf{label}(neg) \leftarrow F\_e_2(false).$
$\mathsf{label}(pos) \leftarrow ev_3(e_2), F\_e_1(true), U\_e_1\_e_0(true).$ $\mathsf{label}(neg) \leftarrow F\_e_0(false).$
... ...

The resulting $\mathcal{DM}$VLP $P_{\mathcal{O}}(T')$ contains 824 rules, divided into 735 with $\mathsf{label}(pos)$ and 89 with $\mathsf{label}(neg)$.

In Example 3, most rules are again obscure consequences of the real property. But some rules are explicit: $\mathsf{label}(neg) \leftarrow F\_e_2(false)$ and $\mathsf{label}(neg) \leftarrow F\_e_0(false)$, state that both e_0 and e_2 must be present in a positive sequence.

3.3 Encoding Complex LTL Properties

LTL allows to model interesting temporal patterns as shown in [1] where they study infinity sensibility of some specific LTL formula. Table 1 shows some examples of these properties. Encoded as new variables, these properties can be used to enhance the explainability of the rules learned in our running example. Using the encoding of Example 3 and the 18 properties considered in [1] is not enough to construct a rule that explains all positive examples of Example 2. Here, we need to consider an additional property, the "not precedence": $G(b \implies \neg F(a))$, i.e., a cannot appear before b.

Example 4. Using these properties and **GULA** as in Example 3, we obtain:
$\mathsf{label}(pos) \leftarrow existence\_e_0(True), existence\_e_2(True), not\_precedence\_e_2\_e_0(True).$
$\mathsf{label}(pos) \leftarrow not\_precedence\_e_0\_e_2(False), not\_precedence\_e_2\_e_0(True).$

Property	LTL formula	Description
Existence	$F(a)$	a must appear at least once
Absence 2	$\neg F(a \wedge F(a))$	a can appear at most once
Choice	$F(a) \vee F(b))$	a or b must appear
Exclusive choice	$(F(a) \vee F(b)) \wedge \neg(F(a) \wedge F(b))$	Either a or b must appear, but not both
Resp. existence	$F(a) \implies F(b)$	if a appear, then b must appear as well
Coexistence	$(F(a) \implies F(b)) \wedge (F(b) \implies F(a))$	Either a and b both appear, or none of them
Response	$G(a \implies F(b))$	Every time a appears, b must appear afterwards
Precedence	$\neg(U(a,b) \vee G(a))$	b can appear only if a appeared before
Not coexistence	$\neg(F(a) \wedge F(b))$	Only one among a and b can appear, but not both

Table 1: Examples of sequence properties from [1].

$\mathsf{label}(pos) \leftarrow existence\_e_2(True), not\_precedence\_e_2\_e_0(True), \mathsf{ev}_0(e_0).$
$\mathsf{label}(pos) \leftarrow existence\_e_0(True), not\_precedence\_e_2\_e_0(True), \mathsf{ev}_3(e_2).$
...

In Example 4, the first rule is the exact representation of the function applied to generate the example, which was: $F(e_0) \wedge F(e_2) \wedge G(e_2 \implies \neg F(e_0))$. The second rule uses $not\_precedence\_e_0\_e_2(False)$ as a divert way to ensure the existence of both e_0 and e_2. The two other rules make use of explicit positions to get the existence of either e_0 or e_2.

3.4 Discussion

In Example 4, we see a few examples of rules that could be discarded. Indeed, **GULA** learns many rules that are redundant when the meaning of the property is known (which **GULA** is oblivious of). Given a subsumption relationship between the encoded properties, a post-processing of the learned rules could be done to simplify or discard rules. Furthermore, in these examples, we guided the rule learning by only giving the property of interest to **GULA** ("existence" and "not precedence") and the optimal program is already almost a thousand rules: 801 $\mathsf{label}(pos)$ rules and 108 $\mathsf{label}(neg)$ rules. The rules shown in Example 4 can be found by weighting and ordering rules according to the number of examples they match. The two first rules are the only ones matching all 17 positive sequences of Example 2. If given all 18 properties of [1] and the "not precedence" property, the optimal program will, in theory, still contain the rules shown in Example 4 plus many others. But it would require to handle more than 100 variables to do so, which is too much for **GULA** to handle in reasonable time.

In practice, it is more interesting to use **PRIDE** [9], **GULA**'s polynomial approximated version, to explore the search space in reasonable time. Although **PRIDE** outputs a subset of the optimal program and thus can miss interesting rules, it can be given some guidance in the form of heuristics, such as variable ordering, to find those "best" rules we are interested in here. All the examples of this paper have been generated using the open source python package pylfit[6] and are available as a Jupiter notebook[7] on the pylfit Github repository.

[6] Package pylfit source code is available at: `https://github.com/Tony-sama/pylfit/`
[7] Case study notebook: `https://github.com/Tony-sama/pylfit/blob/master/tests/evaluations/ilp2022/lfit-sequence-patern-learning.ipynb`

4 Conclusion

In this paper, we proposed an extension of LFIT theory that allows to encode properties of transitions as additional variables, allowing **GULA** to learn rules that exploit them. We proposed a modeling of event sequences and their temporal properties allowing to use **GULA** to learn rules combining properties to explain sequences of interest. Being able to include properties of transitions in the learning process can be useful in a various range of application fields. For instance, in biology, some information on the dynamics of the system to be modelled is expressed as a LTL property by modelers. Inclusion of such knowledge in the global learning process can give more expressive rules about the dynamics and lead to a better understanding of the studied systems by the biologists. We showed through a case study that such encoding can indeed allow to learn more meaningful rules and to capture complex temporal patterns.

However, by encoding properties, we increase the number of variables considered, which leads to a combinatorial explosion of the run time for **GULA**. Its polynomial approximated version **PRIDE** would be preferred in practice, with additional heuristics allowing to guide its search towards comprehensive rules.

References

1. De Giacomo, G., De Masellis, R., Montali, M.: Reasoning on LTL on finite traces: Insensitivity to infiniteness. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 28 (2014)
2. Inoue, K., Ribeiro, T., Sakama, C.: Learning from interpretation transition. Machine Learning **94**(1), 51–79 (2014)
3. Katzouris, N., Artikis, A., Paliouras, G.: Incremental learning of event definitions with inductive logic programming. Machine Learning **100**(2), 555–585 (2015)
4. Muggleton, S.: Inductive logic programming: derivations, successes and shortcomings. ACM SIGART Bulletin **5**(1), 5–11 (1994)
5. Pencolé, Y., Subias, A.: Diagnosability of event patterns in safe labeled time petri nets: a model-checking approach. IEEE Transactions on Automation Science and Engineering (2021)
6. Petri, C.A.: Kommunikation mit Automaten. Ph.D. thesis, Fakultät für Mathematik und Physik, Technische Hochschule Darmstadt, Darmstadt (Allemagne) (1962)
7. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977). pp. 46–57 (1977)
8. Ribeiro, T., Folschette, M., Magnin, M., Inoue, K.: Learning any memory-less discrete semantics for dynamical systems represented by logic programs. Machine Learning (2021)
9. Ribeiro, T., Folschette, M., Magnin, M., Inoue, K.: Polynomial algorithm for learning from interpretation transition. In: 1st International Joint Conference on Learning & Reasoning. pp. 1–5 (2021)
10. Sato, S., Watanabe, Y., Seki, H., Ishii, Y., Yuen, S.: Fault diagnosis for distributed cooperative system using inductive logic programming. In: 2020 IEEE International Conference on Prognostics and Health Management (ICPHM). pp. 1–8. IEEE (2020)