Inductive Learning from State Transitions over Continuous Domains

Tony Ribeiro, Sophie Tourret, Maxime Folschette, Morgan Magnin, Domenico Borzacchiello, Francisco Chinesta, Olivier Roux, Katsumi Inoue

Inductive Learning from State Transitions over Continuous Domains

Tony Ribeiro[1], Sophie Tourret[2], Maxime Folschette[3], Morgan Magnin[1], Domenico Borzacchiello[5], Francisco Chinesta[6], Olivier Roux[1], and Katsumi Inoue[4]

[1] Laboratoire des Sciences du Numérique de Nantes (LS2N), 1 rue de la Noë, 44321 Nantes, France
tony_ribeiro@irccyn.ec-nantes.fr,
[2] Max-Planck-Institut für Informatik, Saarland Informatics Campus, 66123 Saarbrücken, Germany
[3] Univ Rennes, Inria, CNRS, IRISA, IRSET, F-35000 Rennes, France
[4] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
[5] Institut de Calcul Intensif, 1 rue de la Noë, 44321 Nantes, France
[6] PIMM, ENSAM ParisTech, 151 Boulevard de l'Hopital, 75013 Paris, France

Abstract. Learning from interpretation transition *(LFIT)* automatically constructs a model of the dynamics of a system from the observation of its state transitions. So far, the systems that *LFIT* handles are restricted to discrete variables or suppose a discretization of continuous data. However, when working with real data, the discretization choices are critical for the quality of the model learned by *LFIT*. In this paper, we focus on a method that learns the dynamics of the system directly from continuous time-series data. For this purpose, we propose a modeling of continuous dynamics by logic programs composed of rules whose conditions and conclusions represent continuums of values.

Keywords: Continuous Logic Programming, Learning From Interpretation Transition, Dynamical Systems, Inductive Logic Programming

1 Introduction

Learning the dynamics of systems with many interactive components becomes more and more important due to many applications, e.g., multi-agent systems, robotics and bioinformatics. Knowledge of system dynamics can be used by agents and robots for planning and scheduling. In bioinformatics, learning the dynamics of biological systems can correspond to the identification of the influence of genes and can help to understand their interactions. Dynamic system modeling based on time-series data can be classified into discrete and continuous approaches. Discrete and logic-based modeling methodologies assume that the temporal evolution of each entity or variable describing the system occurs in synchronous discrete time steps. These methods seek to infer the regulation functions that update the state of each variable based on the states at previous time steps. In contrast with this approach, continuous models are defined by

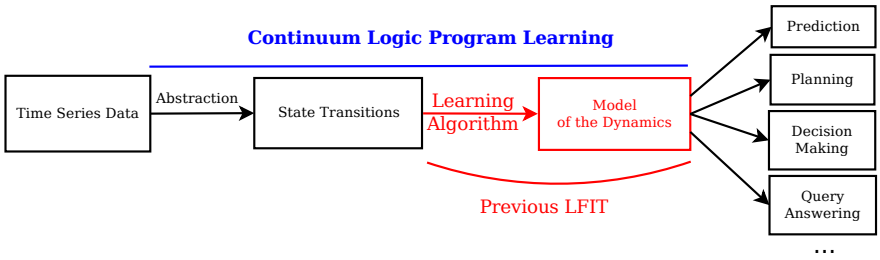


Fig. 1: **Existing work**: assumes a discretization of the time-series data used as input to LFIT. **New method**: no abstraction of the time-series data.

differential equations in which the rate of change of a given variable is related to the actual state of the system. Continuous approaches do not need the discretization of the real-valued measurement data. As a consequence, using real-valued parameters over a continuous timescale yields more reliable results, at least in theory, because it does not introduce any discretization related error. A review of both approaches, outlining their advantages and limitations, with specific applications to gene regulatory networks, is found in [8]. In this paper, we propose a logic-based modeling, which like continuous approaches, allows to deal with real-valued measured data. It is however assuming discrete time steps.

Learning from interpretation transition *(LFIT)* [7] has been proposed to automatically construct a model of the dynamics of a system from the observation of its state transitions. Figure 1 shows this learning process. Given some raw data, like time-series data 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, different inference algorithms that model the system as a logic program are proposed. The semantics of system dynamics can differ regarding the synchronism of its variables, the determinism of its evolution and the influence of its history. The LFIT framework proposes several modeling and learning algorithms to tackle those different semantics. So far the following systems are tackled: memory-less synchronous consistent systems [7], systems with memory [13], non-consistent systems [10].

So far, the discretization of the raw data was assumed given. Its quality is critical to obtain good quality models out of LFIT. For example when learning gene expression evolution, it means that the gene expression levels must be known beforehand. This information is usually unknown and statistical methods are used to guess it. Those methods rely most of the time on static state analysis to provide a division of the gene expressions into meaningful intervals [9,6]. But the levels of expressions and the dynamics are inseparable in biological models. Sometimes those levels of expressions are in fact what must be learned.

In this paper, we learn both at the same time (see Figure 1). For this purpose, we extend LFIT to handle variables representing intervals instead of singletons. In the case of learning gene expression levels, a method that builds dynamical Bayesian networks has been proposed in [11], but it provides no theoretical guar-

antees. In contrast, our approach, that computes continuous logic programs, is generic and comes with soundness and completeness results. Time series data are considered in [14] but continuous data are statically discretized into interval predicate representing statistical properties and their goal is classification. The best-known logic handling intervals is temporal logic [4], which is solely concerned with temporal intervals. In [3] learning of temporal interval relationship defined by [1] (like meet or overlap) is considered. We rely on some of those relationships for rule comparison but learning those relations is not our concern. Temporal logic aside, the other related techniques focus on applying continuous functions on them [12]. The closest to our approach is interval constraint programming [2], but it handles static equational systems. All these techniques are thus unsuitable to solve the problem considered here, where the time is discrete, the values continuous and no continuous function is required.

The organization of the paper is as follows. Section 2 provides a formalization of continuum logic programs, the learning operations and their properties. Section 3 presents the ACEDIA learning algorithm and its experimental evaluation.

2 Continuum Logic and Program Learning

In this section, the concepts necessary to understand the learning algorithm are formalized. In Sect. 2.1 the basic notions of *continuum logic* ($\mathcal{C}$L) and a number of important properties that the learned programs must have are presented. Then in Sect. 2.2 the operations that are performed during the learning, as well as results about the preservation of the properties introduced in Sect. 2.1 throughout the learning are exposed.

2.1 Continuum Logic Programs

Let $\mathcal{V} = \{\mathrm{v}_1, \ldots, \mathrm{v}_n\}$ be a finite set of n variables and $\mathcal{I}_\mathbb{R}$ be the set of all intervals in $\mathbb{R}$. We use basic interval arithmetic operations such as intersection, hull and avoid. Formally for $I_1, I_2 \in \mathcal{I}_\mathbb{R}$, $I_1 \cap I_2 = \{x \in \mathbb{R} \mid x \in I_1 \wedge x \in I_2\}$,

$$\mathrm{hull}(I_1, I_2) = \begin{cases} I_1 \text{ if } I_2 = \emptyset, \\ I_2 \text{ if } I_1 = \emptyset, \\ \{x \in \mathbb{R} \mid \exists y \in I_1, \exists z \in I_2, y \leq x \leq z \vee z \leq x \leq y\} \text{ otherwise.} \end{cases}$$

and $\mathrm{avoid}(I_1, I_2) = \{\{x \in I_1 \mid \forall x' \in I_2, x < x'\}, \{x \in I_1 \mid \forall x' \in I_2, x > x'\}\}$.

The atoms of $\mathcal{C}$L are of the form v^I where $\mathrm{v} \in \mathcal{V}$ and $I \in \mathcal{I}_\mathbb{R}$. An atom v^I is *unit* when $I = \{x\}$ and *empty* when $I = \emptyset$. A $\mathcal{C}$L rule is defined by:

$$R = \mathrm{v}^I \leftarrow \mathrm{v}_1^{I_1} \wedge \cdots \wedge \mathrm{v}_n^{I_n} \tag{1}$$

where v^I and $\mathrm{v}_i^{I_i}$ for $1 \leq i \leq n$ are atoms in $\mathcal{C}$L. The atom on the left-hand side of the arrow is called the *head* of R and is denoted $h(R)$. The notation $\mathrm{v}_{h(R)}$ denotes the variable that occurs in $h(R)$. The conjunction on the right-hand side of the arrow is called the *body* of R, written $b(R)$. The conjunction $b(R)$,

that contains a single occurrence of each variable in $\mathcal{V}$, is assimilated to the set $\{\mathrm{v}_1^{I_1}, \ldots, \mathrm{v}_n^{I_n}\}$ and we use set operations such as $\in$ and $\cap$ on it. A *continuum logic program* ($\mathcal{C}$LP) is a set of $\mathcal{C}$L rules. Intuitively, the rule R has the following meaning: the variable v takes a value in I at the next step if each variable v_i takes a value in I_i at the current step.

The two following definitions introduce relations between atoms and between rules that are used further along.

Definition 1 (Relations between atoms). *Two atoms $a = \mathrm{v}^I$ and $a' = \mathrm{v}^{I'}$ that are based on the same variable $\mathrm{v} \in \mathcal{V}$ can have the following relationships with each other:*

- *a and a'* overlap *when $I \cap I' \neq \emptyset$, written $a \sqcap a'$,*
- *a* subsumes *a' when $I' \subseteq I$, written $a' \sqsubseteq a$.*

In the last case, we also write that a is more general *than a' (resp. a' is* more specific *than a). The notion of* subsumption *is straightforwardly extended to conjunctions of atoms B_1 and B_2 in the following way. B_1 subsumes B_2, written $B_2 \sqsubseteq B_1$ iff:*

$$\forall a \in B_1, \exists a' \in B_2, \text{ such that } a' \sqsubseteq a.$$

Definition 2 (Rules Domination). *Let R_1, R_2 be two $\mathcal{C}$L rules. The rule R_1* dominates *R_2, written $R_2 \leq R_1$ if $h(R_1) \sqsubseteq h(R_2)$ and $b(R_2) \sqsubseteq b(R_1)$.*

Proposition 1. *Let R_1, R_2 be two $\mathcal{C}$L rules. If $R_1 \leq R_2$ and $R_2 \leq R_1$ then $R_1 = R_2$.*

Proof. Let R_1, R_2 be two $\mathcal{C}$L rules such that $R_1 \leq R_2$ and $R_2 \leq R_1$. Then $h(R_1) \sqsubseteq h(R_2)$ and $h(R_2) \sqsubseteq h(R_1)$, hence $h(R_1) = \mathrm{v}^{I_1}$ and $h(R_2) = \mathrm{v}^{I_2}$ and $I_1 \subseteq I_2 \subseteq I_1$ thus $I_1 = I_2$ and $h(R_1) = h(R_2)$. The same reasoning is applied on each variable to conclude $b(R_1) = b(R_2)$. □

Rules with more specific heads and more general bodies dominate the other rules. In practice, these are the rules we are interested in since they cover the most general cases and give the most accurate results.

The dynamical system that we want to learn the rules of is represented by a succession of *continuum states* as formally defined below.

Definition 3 (Continuum State). *A* continuum state *s is a function from $\mathcal{V}$ to $\mathbb{R}$, i.e. it associates a real value to each variable in $\mathcal{V}$. It represents the set of unit atoms $\{\mathrm{v}_1^{\{x_1\}}, \ldots, \mathrm{v}_n^{\{x_n\}}\}$. We write $\mathcal{S}$ to denote the set of all continuum states and a pair of states $(s, s') \in \mathcal{S}^2$ is called a* transition.

The following definitions and propositions describe the interactions between states and rules.

Definition 4 (Rule-states matching). *Let $s \in \mathcal{S}$. The $\mathcal{C}$L rule R* matches *s, written $R \sqcap s$, if $\forall a \in b(R)$, $\exists a' \in s$ such that $a \sqcap a'$.*

A rule is activated only when it matches the current state of the system.

Definition 5 (Cross-matching). *Let R and R' be two $\mathcal{C}$L rules. These rules* cross-match*, written $R \sqcap R'$ when there exists $s \in \mathcal{S}$ such that $R \sqcap s$ and $R' \sqcap s$.*

Cross-matching can also be defined without the use of a matching state.

Proposition 2 (Cross-matching). *Let R and R' be two $\mathcal{C}$L rules.*

$$R \sqcap R' \text{ iff } \forall (\mathrm{v}^I, \mathrm{v}^{I'}) \in b(R) \times b(R'), \mathrm{v}^I \sqcap \mathrm{v}^{I'}.$$

Proof. For the direct implication, assume given two $\mathcal{C}$L rules R and R' such that $R \sqcap R'$. By definition, there exists $s \in \mathcal{S}$ such that s matches both R and R'. Also by definition, for all $(\mathrm{v}^I, \mathrm{v}^{I'}) \in b(R) \times b(R')$, there exists $a, a' \in s$ such that $\mathrm{v}^I \sqcap a$ and $\mathrm{v}^{I'} \sqcap a'$. Moreover, by the definition of a state, $a = a' = \mathrm{v}^{\{x\}}$. Thus $x \in (I \cap I')$, hence $\mathrm{v}^I \sqcap \mathrm{v}^{I'}$.

For the converse implication, it suffices to construct a suitable $s \in \mathcal{S}$, which can be done in the following way: For all $\mathrm{v} \in \mathcal{V}$, there exists $x_\mathrm{v} \in I \cap I'$, where $\mathrm{v}^I \in b(R)$ and $\mathrm{v}^{I'} \in b(R')$ since by Def. 1, $I \cap I' \neq \emptyset$. The state $s = \{x_\mathrm{v} \mid \mathrm{v} \in \mathcal{V}\}$ is such that $s \sqcap R$ and $s \sqcap R'$. □

The final program we want to learn should be complete and consistent within itself and with the observed transitions. The following definitions formalize these desired properties.

Definition 6 (Rule and program realization). *Let R be a $\mathcal{C}$L rule and $(s, s') \in \mathcal{S}^2$. The rule R* realizes *the transition (s, s'), written $s \xrightarrow{R} s'$, if $R \sqcap s$ and there exists $a \in s'$ such that $a \sqsubseteq h(R)$. It* realizes *a set of transitions $T \subseteq \mathcal{S}^2$, written $\overset{R}{\hookrightarrow} T$ if for all $(s, s') \in T, s \xrightarrow{R} s'$.*

A $\mathcal{C}$LP P realizes *(s, s'), written $s \xrightarrow{P} s'$, if for all $\mathrm{v} \in \mathcal{V}$, there exists $R \in P$, $s \xrightarrow{R} s'$. It* realizes *T, written $\overset{P}{\hookrightarrow} T$ if for all $(s, s') \in T$ and all $\mathrm{v} \in \mathcal{V}$, there exists $R \in P$, such that $\mathrm{v}_{h(R)} = \mathrm{v}$ and $s \xrightarrow{R} s'$.*

Definition 7 (Conflicts). Conflicts *can occur between a $\mathcal{C}$L rule R and a state $(s, s') \in \mathcal{S}^2$ or between two $\mathcal{C}$L rules R and R'. The first kind of conflict is when $R \sqcap s$ and $s \overset{R}{\not\rightarrow} s'$ and the second when $\mathrm{v}_{h(R)} = \mathrm{v}_{h(R')}$, $R \sqcap R'$ and neither $h(R) \sqsubseteq h(R')$ or $h(R') \sqsubseteq h(R)$.*

Definition 8 (Consistent program). *A $\mathcal{C}$LP P is* strongly consistent *if it does not contain conflicting rules, i.e. for all $R, R' \in P$ such that $\mathrm{v}_{h(R)} = \mathrm{v}_{h(R')}$ and $R \sqcap R'$, either $h(R) \sqsubseteq h(R')$ or $h(R') \sqsubseteq h(R)$. It is* consistent *when for all conflicting $R, R' \in P$, the rule $R'' = \mathrm{v}^{\emptyset}_{h(R)} \leftarrow \{\mathrm{v}^{I''} \mid \mathrm{v} \in \mathcal{V}, \mathrm{v}^I \in b(R), \mathrm{v}^{I'} \in b(R')$ and $I \cap I' \subseteq I''\}$ belongs to P. Otherwise P is said to be* non-consistent*.*

Note that in the definition of a consistent $\mathcal{C}$LP, due to the conflict between R and R', $I \cap I'$ is never empty. In case there is a blind spot in the observed transitions close to a frontier in the behavior of the system (which always happens to some degree due to the continuous nature of the rules and the discreet nature of the observed transitions) the rules with empty heads indicate the uncertainty of the behavior between the two closest observations.

Definition 9 (Complete program). *A* $\mathcal{C}$LP P *is* complete *if for all* $s \in \mathcal{S}$ *and all* $\mathrm{v} \in \mathcal{V}$ *there exists* $R \in P$ *such that* $R \sqcap s$ *and* $\mathrm{v}_{h(R)} = v$.

Example 1. Let $\mathcal{V} = \{\mathrm{v}_1, \mathrm{v}_2\}$ and consider the two rules $R_1 = \mathrm{v}_1^{[5;8]} \leftarrow \mathrm{v}_1^{]-\infty;\infty[} \wedge \mathrm{v}_2^{]0;5[}$ and $R_2 = \mathrm{v}_1^{\{7\}} \leftarrow \mathrm{v}_1^{]-\infty;\infty[} \wedge \mathrm{v}_2^{[4;9]}$. The rules R_1 and R_2 cross-match but they do not conflict since $\mathrm{v}_{h(R_2)} \sqsubseteq \mathrm{v}_{h(R_1)}$ and they do not dominate each other since $b(R_1) \not\sqsubseteq b(R_2)$ and $b(R_2) \not\sqsubseteq b(R_1)$. They both realize the transition $t = ((10; 4.5), (7; 1))$, however the program $P = \{R_1, R_2\}$ does not realize t because it contains no rule with v_2 as its head variable. P is also not complete, while the $\mathcal{C}$LP $P' = \{\mathrm{v}_1^{[1;2]} \leftarrow \mathrm{v}_1^{]-\infty,\infty[} \wedge \mathrm{v}_2^{]-\infty,\infty[}, \mathrm{v}_2^{\emptyset} \leftarrow \mathrm{v}_1^{]-\infty,\infty[} \wedge \mathrm{v}_2^{]-\infty,\infty[}\}$ is complete.

2.2 Learning operations

The three following definitions describe formally the main operation performed by the learning algorithm, which is to adapt a $\mathcal{C}$LP to realize a new transition with a minimal amount of changes in the dynamics of the program.

Definition 10 (Rule least specialization). *Let* R *be a* $\mathcal{C}$L *rule and* $(s, s') \in \mathcal{S}^2$ *such that* R *and* (s, s') *are conflicting. The* least specialization *of* R *by* (s, s') *is:*

$$P_{\mathrm{spe}}(R, (s, s')) = \bigcup_{\mathrm{v}_s^{\{x\}} \in s} \{h(R) \leftarrow (\{\mathrm{v}_s^{I'}\} \cup b(R) \backslash \{\mathrm{v}_s^{I_s}\})\},$$

where $\mathrm{v}_s^{I_s} \in b(R), I' \in \mathrm{avoid}(I_s, \{x\})$.

Definition 11 (Rule least generalization). *Let* R *be a* $\mathcal{C}$L *rule and* $(s, s') \in \mathcal{S}^2$ *such that* R *and* (s, s') *are conflicting. The* least generalization *of* R *by* (s, s') *is:*

$$P_{\mathrm{gen}}(R, (s, s')) = \{\mathrm{v}^{I''} \leftarrow b(R) \mid h(R) = \mathrm{v}^{I}, \mathrm{v}^{\{x\}} \in s', I'' = \mathrm{hull}(I, \{x\})\}$$

Note that $P_{\mathrm{gen}}(R, (s, s'))$ contains a single rule while the number of rules in $P_{\mathrm{spe}}(R, (s, s'))$ depends on the relationship between the variables in $b(R)$ and in s.

Definition 12 (Rule least revision). *Let* R *be a* $\mathcal{C}$L *rule and* $t \in \mathcal{S}^2$. *The* least revision *of* R *by* t *is:*

$$P_{\mathrm{rev}}(R, t) = \begin{cases} P_{\mathrm{spe}}(R, t) \cup P_{\mathrm{gen}}(R, (s, s')) \text{ when } R \text{ and } t \text{ are conflicting} \\ \{R\} \text{ otherwise.} \end{cases}$$

The least revision *of a* $\mathcal{C}$LP P *by a transition* $t \in \mathcal{S}^2$ *is* $P_{\mathrm{rev}}(P, t) = \bigcup_{R \in P} P_{\mathrm{rev}}(R, t)$.

The intuition behind the least revision is that when a rule is conflicting with a considered transition it is for two possible reasons. Either the conclusion of the rule is correct but the conditions are too general, or the conditions of the rule are correct but the conclusion is too specific.

The following theorem collects properties of the least revision that make it suitable to be used in the learning algorithm.

Theorem 1. *Let R be a $\mathcal{C}$L rule and $(s, s') \in \mathcal{S}^2$. Assume R and (s, s') are conflicting, and let $S_R = \{s'' \in \mathcal{S} \mid R \sqcap s''\}$ and $S_{\text{spe}} = \{s'' \in \mathcal{S} \mid \exists R' \in P_{\text{spe}}(R, (s, s')), R' \sqcap s''\}$. The following results hold:*

1. $S_{\text{spe}} = S_R \setminus \{s\}$,
2. $s \xrightarrow{P_{\text{gen}}(R,(s,s'))} s'$,
3. $P_{\text{rev}}(R, (s, s'))$ *is strongly consistent and contains no rule conflicting with* R *and* (s, s').

Proof.

1. First, let $s'' \in S_R \setminus \{s\}$. Then there exists $\mathrm{v}^{\{x''\}} \in s''$, such that $\mathrm{v}^{\{x\}} \in s$, $\mathrm{v}^I \in b(R)$, $x' \in I$ and $x \neq x'$. Since $x \in I$ because R and (s, s') conflict with each other, we can assume that $x' < x$ (the proof in the case $x' > x$ is symmetrical). Thus, the rule $R' = h(R) \leftarrow b(R) \setminus \{\mathrm{v}^I\} \cup \{\mathrm{v}^{\{x'' \in I | x'' < x\}}\}$ is such that $R' \sqcap s''$ and $R' \in P_{\text{spe}}(R, (s, s'))$ hence $s'' \in S_{\text{spe}}$.
   Now consider $s'' \in S_{\text{spe}}$. By the definition of S_{spe} there exists $R' \in P_{\text{spe}}(R, (s, s'))$ such that $R' \sqcap s''$ thus there exists $I^{\{x\}} \in s''$ such that $\mathrm{v}^I \in b(R)$, $\mathrm{v}^{\{x\}} \in s$ and $x' \in I$, $x' \neq x$. Hence $R \sqcap s''$ but $s'' \neq s$ thus $s'' \in S_R \setminus \{s\}$.
2. Let $P_{\text{gen}}(R, (s, s')) = \{R'\}$. Since (s, s') and R are conflicting, $R \sqcap s$. Moreover given $h(R') = \mathrm{v}^I$, by the definition of P_{gen} and the hull function, there exists $\mathrm{v}^{\{x\}} \in s'$ such that $x \in I = \text{hull}(I, \{x\})$, hence $s \xrightarrow{R'} s'$.
3. Let $R_1, R_2 \in P_{\text{spe}}(R, (s, s'))$. By the definition of $P_{\text{spe}}(R, (s, s'))$, $h(R_1) = h(R_2)$, thus R_1 and R_2 cannot conflict. Now let $R_3 \in P_{\text{gen}}(R, (s, s'))$. Again R_1 and R_3 cannot conflict because $h(R_1) \sqsubseteq h(R_3)$. Thus $P_{\text{rev}}(R, R')$ is free of conflicts. In addition, for all $R' \in P_{\text{spe}}(R, (s, s'))$, $h(R') = h(R)$ and for $\{R'\} = P_{\text{gen}}(R, (s, s'))$, $h(R) \sqsubseteq h(R')$ by the definition of the hull function. Finally, $P_{\text{rev}}(R, (s, s'))$ does not conflict with (s, s') due to the two previous points of this theorem.

□

Example 2. Let $\mathcal{V} = \{\mathrm{v}_1, \mathrm{v}_2\}$, $R = \mathrm{v}_1^{\{5\}} \leftarrow \mathrm{v}_1^{]-\infty,\infty[} \wedge \mathrm{v}_2^{]-\infty,8[}$ and $t = ((0; 1), (4, 2))$. Then $P_{\text{rev}}(R, t) = \{\mathrm{v}_1^{[4;5]} \leftarrow \mathrm{v}_1^{]-\infty,\infty[} \wedge \mathrm{v}_2^{]-\infty;8[}, \mathrm{v}_1^{\{5\}} \leftarrow \mathrm{v}_1^{]-\infty;0[} \wedge \mathrm{v}_2^{]-\infty;8[}, \mathrm{v}_1^{\{5\}} \leftarrow \mathrm{v}_1^{]0;\infty[} \wedge \mathrm{v}_2^{]-\infty;8[}, \mathrm{v}_1^{\{5\}} \leftarrow \mathrm{v}_1^{]-\infty,\infty[} \wedge \mathrm{v}_2^{]-\infty,1[}, \mathrm{v}_1^{\{5\}} \leftarrow \mathrm{v}_1^{]-\infty,\infty[} \wedge \mathrm{v}_2^{]1,8[}\}$. The first rule in $P_{\text{rev}}(R, t)$ is the least generalization of R.

The following definition groups all the properties that we want the learned program to have.

Definition 13 (Suitable and optimal program). *Let $T \subseteq \mathcal{S}^2$. A $\mathcal{C}$LP P* is suitable for T *when:*

- P *is consistent,*
- P *is complete,*
- P *realizes* T*,*
- *there is no rule with empty head in* P *that matches a* s *such that* $(s, s') \in T$*,*
- *for all* $\mathcal{C}$L *rules* R *not conflicting with a* s *such that* $(s, s') \in T$*, there exists* $R' \in P$ *such that* $R \leq R'$*.*

If in addition, for all $R \in P$, all the $\mathcal{C}$L rules R' belonging to $\mathcal{C}$LP suitable for T are such that $R \leq R'$ implies $R' \leq R$ then P is called optimal.

Proposition 3. *Let $T \subseteq \mathcal{S}^2$. The $\mathcal{C}$LP optimal for T is unique and denoted $P_{\mathcal{O}}(T)$.*

Proof. Let $T \subseteq \mathcal{S}^2$. Assume the existence of two distinct $\mathcal{C}$LPs optimal for T, denoted by $P_{\mathcal{O}_1}(T)$ and $P_{\mathcal{O}_2}(T)$ respectively. Then w.l.o.g. we consider that there exists a $\mathcal{C}$L rule R such that $R \in P_{\mathcal{O}_1}(T)$ and $R \notin P_{\mathcal{O}_2}(T)$. If R is conflicting with T, since $P_{\mathcal{O}_1}(T)$ realizes T there exists $R' \in P_{\mathcal{O}_1}(T)$ and $(s, s') \in T$ such that $R \sqcap s$, $R' \sqcap s$, $s \overset{R}{\not\to} s'$, $s \xrightarrow{R'} s'$ and $\mathrm{v}_{h(R)} = \mathrm{v}_{h(R')}$. Hence R and R' are conflicting, thus there exists $R'' = \mathrm{v}^{\emptyset}_{h(R)} \leftarrow \{\mathrm{v}^{I''} \mid \mathrm{v}^{I} \in b(R), \mathrm{v}^{I'} \in b(R'), I \cap I' \subseteq I''\}$. But then $R'' \sqcap s$ and $P_{\mathcal{O}_1}(T)$ is not suitable for T, a contradiction. Thus R is not conflicting with T and there exists a $\mathcal{C}$L rule $R_2 \in P_{\mathcal{O}_2}(T)$, such that $R \leq R_2$. By the definition of an suitable program, there exists $R_1 \in P_{\mathcal{O}_1}(T)$ such that $R_2 \leq R_1$ since R_2 is not conflicting with T. Thus $R \leq R_1$ and by the definition of an optimal program $R_1 \leq R$. By Proposition 1, $R_1 = R$ and thus $R \leq R_2 \leq R$ hence $R_2 = R$, a contradiction. □

The starting point of the learning algorithm is $P_{\mathcal{O}}(\emptyset)$, described in the following proposition.

Proposition 4. $P_{\mathcal{O}}(\emptyset) = \{\mathrm{v}^{\emptyset} \leftarrow \{\mathrm{v}'^{]-\infty,\infty[} \mid \mathrm{v}' \in \mathcal{V}\} \mid \mathrm{v} \in \mathcal{V}\}$.

Proof. Let $P = \{\mathrm{v}^{\emptyset} \leftarrow \{\mathrm{v}'^{]-\infty,\infty[} \mid \mathrm{v}' \in \mathcal{V}\} \mid \mathrm{v} \in \mathcal{V}\}$. The $\mathcal{C}$LP P is consistent and complete by construction. Like all $\mathcal{C}$LPs, $\overset{\emptyset}{\hookrightarrow} P$ and there is no transition in $\emptyset$ to match with the rules in P. In addition, by construction, the rules of P dominate all $\mathcal{C}$L rules. □

The $\mathcal{C}$LP optimal for a set of transitions can be obtained from any $\mathcal{C}$LP suitable for T by removing all the dominated rules from it, as stated in the following proposition. This means that it suffices to compute a $\mathcal{C}$LP suitable for T to obtain $P_{\mathcal{O}}(T)$ by getting rid of the dominated rules.

Proposition 5. *Let $T \subseteq \mathcal{S}^2$. If P is a $\mathcal{C}$LP suitable for T, then $P_{\mathcal{O}}(T) = \{R \in P \mid \forall R' \in P, R \leq R'$ implies $R' \leq R\}$*

Proposition 6. *Let P be a consistent $\mathcal{C}$LP and $(s, s') \in \mathcal{S}^2$. The $\mathcal{C}$LP $P_{\mathrm{rev}}(P, (s, s'))$ is consistent.*

Proof. Let us assume there exists two $\mathcal{C}$L rules $R_1, R_2 \in P_{\mathrm{rev}}(P, (s, s'))$. Since by Theorem 1, for all $R \in P$, $P_{\mathrm{rev}}(R, (s, s'))$ is strongly consistent, necessarily there exists two distinct $R'_1, R'_2 \in P$ such that $R_1 \in P_{\mathrm{rev}}(R'_1, (s, s'))$ and $R_2 \in P_{\mathrm{rev}}(R'_2, (s, s'))$. The fact that R_1 and R_2 conflict implies that $\mathrm{v}_{h(R_1)} = \mathrm{v}_{h(R_2)} = v$. It also implies that $R_1 \sqcap R_2$. Whether R_1 and R_2 are obtained by least specialization or least generation, the following relationships hold by construction:

1. $b(R_1) \sqsubseteq b(R'_1)$ and $b(R_2) \sqsubseteq b(R'_2)$,
2. $h(R'_1) \sqsubseteq h(R_1)$ and $h(R'_2) \sqsubseteq h(R_2)$.

Due to point 1, $R'_1 \sqcap R'_2$. Since in addition P is consistent and $\mathrm{v}_{h(R'_1)} = \mathrm{v}_{h(R'_2)}$, either $h(R'_1) \sqsubseteq h(R'_2)$ or $h(R'_2) \sqsubseteq h(R'_1)$. If $\mathrm{v}_{h(R'_1)}$ and $\mathrm{v}_{h(R'_2)}$ are not empty, due to point 2, the same relationship also holds between R_1 and R_2, a contradiction with the fact that there is a conflict between R_1 and R_2. Otherwise, one of $\mathrm{v}_{h(R_1)}$ and $\mathrm{v}_{h(R_2)}$ is empty, thus its least generalization's head is sure to be a singleton. Assume w.l.o.g. that $\mathrm{v}_{h(R'_1)}$ is empty. Since R_1 and R_2 conflict with each other, their heads cannot be empty or subsume each other, thus $\{R_1\} = P_{\mathrm{gen}}(R'_1, (s, s'))$ and $R_2 \in P_{\mathrm{spe}}(R'_2, (s, s'))$. Thus $b(R_2)$ avoids s on a variable v_* and there is a rule $R \in P_{\mathrm{spe}}(R'_1, (s, s'))$ that avoids s on the same variable and in the same way (either over or under it). The rule R has an empty head since R'_1 also has one and for all $\mathrm{v} \in \mathcal{V}$, if $\mathrm{v}^{I_1} \in b(R_1)$, $\mathrm{v}^{I_2} \in b(R_2)$ and $\mathrm{v}^I \in b(R)$ then $I_1 \cap I_2 \subseteq I$ since I coincides with I_1 except on v_* where $I \in \mathrm{avoid}(I_1, \{x\})$ and I overlaps with I_2 from its bound at x and until the bound of I_1, thus covering $I_1 \cap I_2$ entirely. □

The following theorem in association with the three previous results gives a method to iteratively compute $P_{\mathcal{O}}(T)$ for any $T \subseteq \mathcal{S}^2$, starting from $P_{\mathcal{O}}(\emptyset)$.

Theorem 2. *Let $T \subseteq \mathcal{S}^2$ and $(s, s') \in \mathcal{S}^2$. $P_{\mathrm{rev}}(P_{\mathcal{O}}(T), (s, s'))$ is a $\mathcal{C}$LP suitable for $T \cup \{(s, s')\}$.*

Proof.

Let $P = P_{\mathrm{rev}}(P_{\mathcal{O}}(T), (s, s'))$. Since $P_{\mathcal{O}}(T)$ is consistent, by Prop. 6, P is also consistent. Since $P_{\mathcal{O}}(T)$ is complete, by the two first points of Th. 1, P is also complete. By Th. 1, P is not in conflict with the rules of $P_{\mathcal{O}}(T)$, and since P is also complete, $\overset{P}{\hookrightarrow} T$. In addition, since $P_{\mathcal{O}}(T)$ is complete, for each $\mathrm{v} \in \mathcal{V}$, there exists a $\mathcal{C}$L rule $R \in P_{\mathcal{O}}(T)$ such that $\mathrm{v}_{h(R)} = \mathrm{v}$ and $R \sqcap s$. By Th. 1, it means that for each of these rules, $s \xrightarrow{P_{\mathrm{gen}}(R,(s,s'))} s'$, hence $s \xrightarrow{P} s'$ and $\overset{P}{\hookrightarrow} (T \cup \{(s, s')\})$. Assume the existence of a rule $R \in P$ with empty head and matching a state s'' where $(s'', s''') = t \in T \cup \{(s, s')\}$. If $R \in P_{\mathcal{O}}(T)$ then $t = (s, s')$ or $P_{\mathcal{O}}(T)$ is not suitable for T. In this case, since $R \sqcap t$ and R has not been revised, $s \xrightarrow{R} s'$, a contradiction with the emptiness of the head of R. Otherwise, there exists a $\mathcal{C}$L rule $R' \in P_{\mathcal{O}}(T)$ such that $R \in P_{\mathrm{spe}}(R', t)$ because a generalization cannot produce rules with empty heads. Then by Th. 1, $t \neq (s, s')$ and since $R \sqcap s''$, we also have $R' \sqcap s''$ by the definition of the specialization operation. For the same reason, the head of R' is empty. Thus, $P_{\mathcal{O}}(T)$ is not suitable for T, a contradiction. To prove that P verifies the last point of the definition of a suitable $\mathcal{C}$LP, let R be a $\mathcal{C}$L rule not conflicting with $T \cup \{(s, s')\}$. Since R is also not conflicting with T, there exists $R' \in P_{\mathcal{O}}(T)$ such that $R \leq R'$. If R' is not conflicting with (s, s'), then $R' \in P$. Otherwise, $R \leq R'$ and R' is in conflict with (s, s') (but R is not). Thus there exists at least one variable $\mathrm{v} \in \mathcal{V}$ such that $\mathrm{v}^I \in b(R)$, $\mathrm{v}^{I'} \in b(R')$, $\mathrm{v}^{\{x\}} \in s$ $x \in I'$, $x \notin I$ and $I \subseteq I'$. Then one of the intervals in $\mathrm{avoid}(I', \{x\})$ contains I by the definition of the function avoid. Let us denote this interval by I''. The rule $R'' \in P_{\mathrm{spe}}(R', (s, s'))$ such that $\mathrm{v}^{I''} \in b(R'')$ verifies $R \leq R''$ because $\mathrm{v}^I \sqsubset \mathrm{v}^{I''}$ and R'' coincides with R' on all other body and head variables. □

3 ACEDIA

In this section we present **ACEDIA**, the Abstraction-free Continuum Environment Dynamics Inference Algorithm and its experimental evaluation.

3.1 Algorithm

ACEDIA learns a $\mathcal{C}$LP from time-series data over continuous domains. Those time-series data are observations of a system's state transitions ($\mathcal{S}^2$). Given as input a set of transitions $T \subseteq \mathcal{S}^2$, **ACEDIA** iteratively constructs a model of the system by applying the method formalized in the previous section to compute $P_{\mathcal{O}}(T)$ as follows:

ACEDIA: Abstraction-free Continuum Environment Dynamics Inference Algorithm

- **INPUT:** a set of transitions $T \subseteq \mathcal{S}^2$.
- Initialize P as $P_{\mathcal{O}}(\emptyset)$.
- For each transition $(s, s') \in T$
  - Extract each rule R of P that conflicts with (s, s').
  - For each rule R
    - Compute its least revision $P' = P_{\text{rev}}(R, (s, s'))$.
    - Remove all the rules in P' dominated by a rule in P or P'.
    - Remove all the rules in P dominated by a rule in P'.
    - Add all remaining rules in P' to P.
- **OUTPUT:** $P = P_{\mathcal{O}}(T)$.

Algorithm 1 **ACEDIA**(T)

INPUT: $T \subseteq \mathcal{S}^2$
OUTPUT: $P_{\mathcal{O}}(T)$

// 1) Initialization of P
$P = \emptyset$ // The empty logic program
for each $\mathrm{v} \in \mathcal{V}$ **do**
  $P := P \cup \{\mathrm{v}^{\emptyset} \leftarrow \{\mathrm{v}'^{]-\infty,\infty[} \mid \mathrm{v}' \in \mathcal{V}\}\}$

// 2) Revision of P for each transition
for each $(s, s') \in T$ **do** :
  $conflicts := \emptyset$
  //2.1) Extraction of conflicting rules
  for each $R \in P$ **do**
    if $b(R)$ conflicts with (s, s') **then**
      $P := P \setminus \{R\}$
      $conflicts := conflicts \cup \{R\}$
  //2.2) Revision of conflicting rules
  for each $R \in conflicts$ **do**
    $LR :=$ **least_revision**$(R, (s, s'))$
    for each $R' \in LR$ **do**
      if $\nexists R'' \in P, R' \leq R''$ **then**
        $P := P \setminus \{R''' \in P \mid R''' \leq R'\} \cup \{R'\}$
return P

Algorithm 2 **least_revision**(R, t)

INPUT: a rule R and a transition $t = (s, s')$,
OUTPUT: $LR = P_{\text{rev}}(R, t)$.

// 1) Least generalization
$x = I, \mathrm{v}^I \in s'$
if $h(R) = \mathrm{v}^{\emptyset}$ **then**
  $h = \mathrm{v}^{\{x\}}$
else
  if $x < min(h(R))$ **then**
    $h := \mathrm{v}^{[x, max(h(R))]}$
  else
    $h := \mathrm{v}^{[min(h(R)), x]}$
$LR := \{h \leftarrow b(R)\}$

// 2) Least specialization
for each $\mathrm{v}^I \in b(R)$ **do**
  $R_{min} :=$
$h(R) \leftarrow (b(R) \setminus \{\mathrm{v}^I\} \cup \{\mathrm{v}^{]min(I), x[} | min(I) \neq x\})$
  $R_{max} :=$
$h(R) \leftarrow (b(R) \setminus \{\mathrm{v}^I\} \cup \{\mathrm{v}^{]x, max(I)[} | max(I) \neq x\})$
  $LR := LR \cup \{R_{min}, R_{max}\}$
return LR

Algorithms 1 and 2 provide the detailed pseudocode of the algorithm. Lines 3-6 of Alg. 1 realize the computation of $P_{\mathcal{O}}(\emptyset)$ as defined in Prop. 4. Then the learning is performed iteratively on each transition $t \in T$ by applying the least-revision operation (lines 7-17) as defined in Def. 12 and removing dominated rules (lines 18-20) to ensure the optimality of the obtained program as stated in Prop. 5. The rules obtained by specialization that have empty intervals in their bodies are discarded since they cannot match or realize anything. Another noteworthy difference between the theory and the implementation is that the variables are not necessarily defined over all of $\mathbb{R}$ but may be constrained over an interval in $\mathcal{I}_{\mathbb{R}}$ encompassing all the values associated to each variable in the observed transitions. This does not impact the theoretical results stated below.

Theorem 3 (ACEDIA Termination, soundness, completeness, optimality). *Let $T \subseteq \mathcal{S}^2$. The call ACEDIA($T$) terminates and ACEDIA($T$)=$P_{\mathcal{O}}(T)$*

Proof. Let $T \subseteq \mathcal{S}^2$. The call ACEDIA($T$) terminates because all loops iterate on finite sets.

To prove that ACEDIA(T)=$P_{\mathcal{O}}(T)$, and is thus sound, complete and optimal, it suffices to prove that the main loop (Alg. 1 line 7-20) preserves the invariant $P = P_{\mathcal{O}}(T_i)$ after the i^{th} iteration where T_i is the set of transitions already selected line 8 after the i^{th} iteration for all i from 0 to $|T|$.

Lines 3-6 initialize P to $\{\mathrm{v}^{\emptyset} \leftarrow \{\mathrm{v}'^{]-\infty,\infty[} \mid \mathrm{v}' \in \mathcal{V}\} \mid \mathrm{v} \in \mathcal{V}\}$. Thus by Prop. 4, after line 6, $P = P_{\mathcal{O}}(\emptyset)$.

Let us assume that before the $i+1^{\text{th}}$ iteration of the main loop, $P = P_{\mathcal{O}}(T_i)$. Through the loop lines 11-14, $P' = \{R \in P_{\mathcal{O}}(T_i) \mid R \text{ does not conflict with } (s, s')\}$ is computed. Then the set $P'' = \bigcup\{P_{\text{rev}}(R, (s, s')) \mid R \in P_{\mathcal{O}}(T_i) \backslash P'\}$ is iteratively build through the calls to **least_revision** line 17 and the dominated rules are pruned as they are detected by the loop lines 18-20. Thus by Th. 2 and Prop. 5, $P = P_{\mathcal{O}}(T_{i+1})$ after the $i + 1^{\text{th}}$ iteration of the main loop. □

Theorem 4 (ACEDIA Complexity). *Let $T \subseteq \mathcal{S}^2$ be a set of transitions and $|\mathcal{V}| = n$. The worst-case time complexity of* **ACEDIA** *when learning from T belongs to $\mathcal{O}(|T|^{2n} \times n^5)$ and its worst-case memory use belongs to $\mathcal{O}(|T|^{2n} \times n^2)$.*

Proof. Let x_i be the different values a variable v_i takes in T. Each x_i can be the minimum or maximum value of a continuum. After the initialization of P there is a rule with empty head for each x_i. According to the definition of the least generalization (Def. 11), those continuum will only be extended to hull a x_i, i.e. their min/max will always be a x_i. Thus the number of possible head continuums of a learned rule is $1 + |x_i|^2$ which belongs to $\mathcal{O}(1 + |T|^2)$. Similarly for the rule body, according to the definition of the least specialization (Def. 10), a continuum in a rule body is only reduced to avoid one of the x_i. The only other possible values are ∞ and $-\infty$, thus the number of possible continuums for each body variable belongs to $\mathcal{O}((|x_i| + 2)^2) \sim \mathcal{O}((|T| + 2)^2)$. The possible bodies of a rule are all the combinations of these continuums, thus belong to $\mathcal{O}(((|T| + 2)^2)^n)$. Hence, the total number of possible rules learned by **ACEDIA** belongs to $\mathcal{O}(n \times (1 + |T|^2) \times ((|T| + 2)^{2n}))$. The heads of rules are

represented by an integer that encodes the variable it refers to and a continuum represented by two real numbers. The body of a rule is a vector of n pairs of variable/continuum encoded in the same way. Thus the memory use of a rule belongs to $\mathcal{O}(3+3\times n)$. **Conclusion:** the memory use of **ACEDIA** belongs to $\mathcal{O}(|P|) \sim \mathcal{O}(n\times((1+|T|^2)\times((|T|+2)^2)^n)\times(3+3\times n))$, i.e. $\mathcal{O}(|P|) \sim \mathcal{O}(|T|^{2n}\times n^2)$.

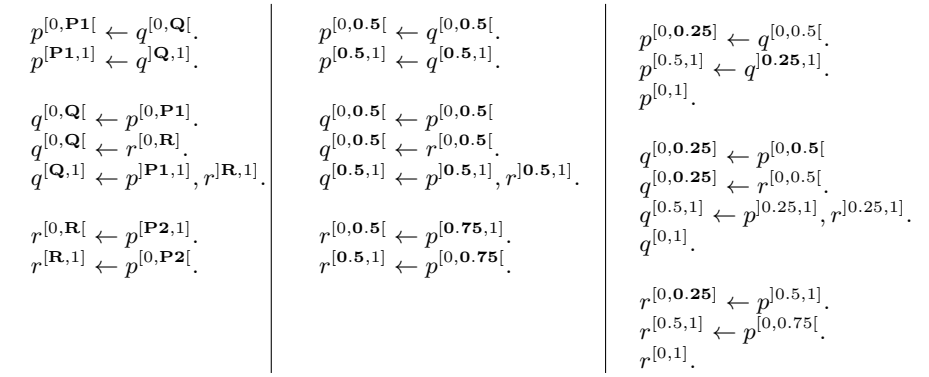
ACEDIA starts with the generation of the logic program $P_{\mathcal{O}}(\emptyset)$, containing n rules. A rule R has one head variable and its body contains each variable: building a rule belongs to $\mathcal{O}(|R|) \sim \mathcal{O}(1+n)$. Thus the initialization of P belongs to $\mathcal{O}(n\times|R|)$. Afterward, the algorithm checks each transition of T iteratively and extracts conflicting rules from P. Checking conflict between two rules requires to compare their heads and all their body atoms: it belongs to $\mathcal{O}(|R|)$. Each rule of P is checked, thus extracting conflicting rules belongs to $\mathcal{O}(|P| \times |R|)$. Each conflicting rule is revised using least revision (Def. 12). This operation generates one rule by least generalization (one head continuum extension) and $2n$ rules by least revision (2 for each atom in the body: one that avoids the value from below and the other that avoids it from the top), it belongs to $\mathcal{O}(|LR|) \sim \mathcal{O}((1+2n)\times|R|)$. Each revision is then compared to the rules of P to check domination (Def. 2). Checking domination requires to compare head continuum and all body continuum: it belongs to $\mathcal{O}(|R|)$. Thus removing the dominated revisions belongs to $\mathcal{O}(|LR|\times|R|\times|P|)$. This is repeated for each transition of T, thus the complete process belongs to $\mathcal{O}(n\times|R|+|T|\times|LR|\times|R|\times|P|) \sim \mathcal{O}(n\times(1+n)+|T|\times(1+2n)\times(1+n)\times(1+n)\times n\times(1+|T|^2)\times(|T|+2)^{2n}\times(3+3\times n)) \sim \mathcal{O}(|T|\times(1+|T|^2)\times(|T|+2)^{2n}\times n^5)$ **Conclusion:** the complexity of **ACEDIA** when learning from T belongs to $\mathcal{O}(|T|^{2n}\times n^5)$. □

3.2 Evaluation

In this section, the benefits from **ACEDIA** are demonstrated on a case study and its scalability is assessed w.r.t. the input size and the number of variables. All experiments were conducted on an Intel Core I7 (6700, 3.4GHz) with 32 Gb of RAM and can be accessed via the hyperlink given in footnote[7].

The first evaluation is a case study on learning a $\mathcal{C}$LP equivalent to a Boolean network of 3 variables. For the purpose of this experiment the levels of expression can be changed by setting the condition/conclusion intervals as shown in Figure 2a and 2b. In the rules body, q and r have a unique expression level but the level of p differs in the dynamics of q and r: to activate q, $p = 0.5$ is enough but $p = 0.75$ is necessary to inhibit r. This is done to show that **ACEDIA** can learn different behaviors and different expression levels for the same variable, while previous versions of **LFIT** assumed the same discretization in all rules. The domain of each variable is restricted to $[0, 1]$, which can be considered like a normalization of the time-series in practice. For readability reasons, in the body of the rules, variables of which the corresponding value is the whole domain are omitted. Considering a precision of 0.25 for each variable value, 150 possible states are generated. The algorithm computes the approximately 1700 possible

[7] Experiments sources: `http://tonyribeiro.fr/data/experiments/ILP_2017.zip`



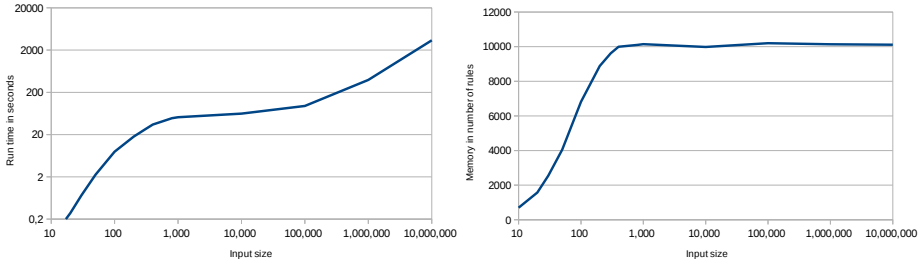
(a) $\mathcal{C}$LP with editable levels of expression in bold.

(b) $\mathcal{C}$LP with levels of expression set to P1=Q=R=0.5 and P2=0.75

(c) **ACEDIA** output from all the transitions of the $\mathcal{C}$LP in Figure 2b

Fig. 2: Experimentation on a $\mathcal{C}$LP with three variables.

transitions according to the $\mathcal{C}$LP. From those transitions, **ACEDIA** learns the original rules to capture the dynamics of the system and finds the expression level of the variables. **ACEDIA**'s output is shown in Figure 2c. The rules are ordered and grouped to follow Figure 2b and the rules with empty heads are omitted. They encode all non-observed states, e.g. $p^{\emptyset} \leftarrow q^{]0,0.25[}$. and $r^{\emptyset} \leftarrow q^{]0.25,0.75[}$. Such minor differences have been highlighted in bold in Figure 2c. The first rule is equivalent to the first rule of p of the original program: the total range of value a variable can take is $[0, 1]$, thus conditions on this continuum can be ignored. By looking closely to the first rule in Figure 2c, it appears to be different from the first rule in Figure 2b: the head of the former is equal to $[0, 0.25]$ while the latter is equal to $[0, 0.5[$. This is as close an approximation as is possible with a precision of 0.25 in the states, and the fact that only closed bounds (respectively open bounds) can be created in the head (respectively body) of a rule. The second rule is equivalent to the second rule of p: here the target continuum $[0.5, 1]$ is approximated by $]0.25, 1]$. The third, as the most general conclusion and conditions, just provides the domain of the variable ; it is an optimal rule of the observations that can be expected to be learned. Here we see that the influence of q over p is correctly learned as well as the level of expression of q to influence p which is 0.5. Regarding the dynamics of q, the rules are equivalent to the three original rules of q if we follow the same reasoning as before. The dynamics of the influences of p and r over q is also learned correctly, as well as the level of expression of p and r. Again line 7 we have a similar rule that just gives the domain of q and can be ignored. The rest of the rules are equivalent to the rules of r of the original program. Here we see that the specific level of expression of p to inhibit r, $P2 = 0.75$ is approximated by the continuum $]0.5, 1]$.



(a) avg. run time (log scale) over 10 runs (b) maximal number of rules

Fig. 3: Evaluation of **ACEDIA**'s scalability w.r.t. input size (log scale) on learning the $\mathcal{C}$LP of Figure 2b.

Variables	run time/rules w.r.t. input transitions ($\|T\|$)						
	$\|T\| = 10$	$\|T\| = 25$	$\|T\| = 50$	$\|T\| = 100$	$\|T\| = 250$	$\|T\| = 500$	$\|T\| = 1000$
2	0.015s / 137	0.031s / 296	0.067s / 425	0.105s / 437	0.143s / 420	0.217s / 422	0.337s / 456
3	0.034s / 464	0.667s / 3K	3.082s / 5K	10s / 8K	31s / 11K	56s / 13K	73s / 13K
4	0.322s / 2K	16s / 14K	127s / 36K	1081s / 86K	T.O.	T.O.	T.O.
5	2.8s / 7K	239s / 58K	4,522s / 203K	T.O.	T.O.	T.O.	T.O.
6	15s / 21K	4,063s / 217K	T.O.	T.O.	T.O.	T.O.	T.O.
7	73s / 45K	22,616s / 596K	T.O.	T.O.	T.O.	T.O.	T.O.
8	424s / 120K	T.O.	T.O.	T.O.	T.O.	T.O.	T.O.
9	2,239s / 228K	T.O	T.O.	T.O.	T.O.	T.O.	T.O.

Table 1: Evaluation of **ACEDIA** scalability over 10 runs: average run time and memory use (in number of rules) on learning mammalian cell Boolean Network evolving the number of variables. Time Out (T.O.) is 10 hours.

This experiment shows that the dynamics of the system and the expression level of each variable are approximated as well as possible by **ACEDIA**.

Figure 3 shows the run time (Figure 3a) and memory use (Figure 3b) of **ACEDIA** on learning the $\mathcal{C}$LP of Figure 2b (middle) with regards to the number of input transitions. In this experiment, the precision of the value of each variable is 0.1 in the interval [0,1], thus there are 11 possibles values for a variable and $11 \times 11 = 121$ possible continuums. In theory, more than 200 millions of rules (121 heads times 121^3 bodies) could be learned, but this number never exceeds much more than $10,000$ at a given time. The domination relation (Def. 2) allows to reduce the number of candidate rules at each step of the learning process. After approximately 400 transitions, the real rules of the system are learned and most of the computation consists in checking those rules against the new transitions and eliminating the remaining rules that are still specific enough to survive. This experiment shows that when the observed system has a small number of variables, the algorithm can be fed with as many observations as wanted.

Table 1 shows the run times and memory use of **ACEDIA** on learning partial mammalian cell Boolean networks [5] by varying the number of variables

considered. The original number of variables is 10. To reduce the variables to $n < 10$, we removed the occurrences of $10 - n$ variables in all original rules, thus creating a new system of n variables. In this experiment the exponential evolution of run time caused by the exponential explosion of the number of generated rules can be seen. For now, **ACEDIA** cannot handle systems with more than 9 variables in a reasonable amount of time and memory when considering 10 transitions and it tends to be limited to 4 variables when more than 10 input transitions are studied. However, as in the previous experiment, we observe the convergence of the number of rules and thus run time for 2 and 3 variables, which hints that such behavior should occur for more variables but with an exponentially greater input size. The current implementation is rather naïve. As with previous **LFIT** algorithms, we can expect better experimental results regarding scalability by developing dedicated data structures and learning heuristics. Such improvements remain as future work.

4 Conclusions

In the previous **LFIT** algorithms, it was assumed that the discretization of raw input data was performed by some third-party agent. Such a hypothesis is rather naive, and does not match with real-life systems since the dynamics of a system is defined by both the levels of expression of variables and their influences on each other. In this paper, we introduce **ACEDIA**, an algorithm to learn the dynamics of a system directly from continuous time-series data. For this purpose we propose a modeling of continuous dynamics by continuum logic programs. As far as we know, this approach is completely new and its strengths and weaknesses are shown through theoretical results and practical evaluations. Similarly to continuous approaches, the modeling we propose allows to deal with real-valued measured data. It however assumes discreet time steps. One of our future works will thus address continuous time dynamics in the **LFIT** framework. It is important to note that this method can also be applied to discrete data like previous **LFIT** algorithms. In the case of multi-valued discrete data, **ACEDIA** learns more compact and expressive rules. Indeed, multiple conditions over different contiguous discrete levels can be expressed by one condition over a continuum including those levels. Where previous **LFIT** algorithms need several rules to express those conditions, **ACEDIA** expresses them with a single one. The detailed comparison of **ACEDIA** with previous **LFIT**s on this kind of data is out of the scope of this paper and remains as a future work.

This paper focuses on the theoretical bases of **ACEDIA** and we are now working on an efficient implementation of the algorithm, with the goal of applying it to real biological time-series data. The complexity of the current algorithm (see Th. 4) limits its current usability to rather small systems as shown by the experimental results. However, the convergences observed gives us good hope about the practical use of the methods.

References

1. J. F. Allen. An interval-based representation of temporal knowledge. In *IJCAI*, volume 81, pages 221–226, 1981.
2. F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget. Revising hull and box consistency. In *Logic Programming: Proceedings of the 1999 International Conference on Logic Programming*, pages 230–244. MIT press, 1999.
3. M. do Carmo Nicoletti, F. O. S. de Sá Lisboa, and E. R. Hruschka. *Learning Temporal Interval Relations Using Inductive Logic Programming*, pages 90–104. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
4. E. A. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, 995(1072):5, 1990.
5. A. Fauré, A. Naldi, C. Chaouiya, and D. Thieffry. Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle. *Bioinformatics*, 22(14):e124–e131, 2006.
6. N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using bayesian networks to analyze expression data. *Journal of computational biology*, 7(3-4):601–620, 2000.
7. K. Inoue, T. Ribeiro, and C. Sakama. Learning from interpretation transition. *Machine Learning*, 94(1):51–79, 2014.
8. G. Karlebach and R. Shamir. Modelling and analysis of gene regulatory networks. *Nature Reviews Molecular Cell Biology*, 9(10):770–780, 2008.
9. S. Y. Kim, S. Imoto, and S. Miyano. Inferring gene networks from time series microarray data using dynamic bayesian networks. *Briefings in bioinformatics*, 4(3):228–235, 2003.
10. D. Martínez Martínez, T. Ribeiro, K. Inoue, G. Alenyà Ribas, and C. Torras. Learning probabilistic action models from interpretation transitions. In *Proceedings of the Technical Communications of the 31st International Conference on Logic Programming (ICLP 2015)*, pages 1–14, 2015.
11. I. Nachman, A. Regev, and N. Friedman. Inferring quantitative models of regulatory networks from expression data. *Bioinformatics*, 20(suppl 1):i248–i256, 2004.
12. M. J. C. Ramon E. Moore, R. Baker Kearfott. *Introduction to interval analysis*. Society for Industrial and Applied Mathematics, 2009.
13. T. Ribeiro, M. Magnin, K. Inoue, and C. Sakama. Learning delayed influences of biological systems. *Frontiers in Bioengineering and Biotechnology*, 2:81, 2015.
14. J. J. Rodríguez, C. J. Alonso, and H. Boström. *Learning First Order Logic Time Series Classifiers: Rules and Boosting*, pages 299–308. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.