Noname manuscript No.
(will be inserted by the editor)

Learning any memory-less discrete semantics for dynamical systems represented by logic programs

Tony Ribeiro · Maxime Folschette · Morgan Magnin · Katsumi Inoue

Received: date / Accepted: date

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 handled were mainly restricted to synchronous deterministic dynamics. However, other dynamics exist in the field of logical modeling, in particular the asynchronous semantics which is widely used to model biological systems. In this paper, we propose a modeling of discrete memory-less multi-valued dynamic systems as logic programs in which a rule represents *what can occur rather than what will occur*. This modeling allows us to represent non-determinism and to propose an extension of LFIT to learn regardless of the update schemes, allowing to capture a large range of semantics. We also propose a second algorithm which is able to learn a whole system dynamics, including its semantics, in the form of a single propositional logic program with constraints. We show through theoretical results the correctness of our approaches. Practical evaluation is performed on benchmarks from biological literature.

Tony Ribeiro
Independant Researcher
Université de Nantes, Centrale Nantes, CNRS, LS2N, F-44000 Nantes, France
National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
E-mail: tony.ribeiro@ls2n.fr,

Maxime Folschette
Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Morgan Magnin
Centrale Nantes, Université de Nantes, CNRS, LS2N, F-44000 Nantes, France
National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan

Katsumi Inoue
National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan

Keywords inductive logic programming · dynamic systems · logical modeling · dynamic semantics

1 Introduction

Learning the dynamics of systems with many interactive components becomes more and more important in many applications such as physics, cellular automata, biochemical systems as well as engineering and artificial intelligence systems. In artificial intelligence systems, knowledge like action rules is employed by agents and robots for planning and scheduling. In biology, learning the dynamics of biological systems corresponds to the identification of influence of genes, signals, proteins and molecules that can help biologists to understand their interactions and biological evolution.

In modeling of dynamical systems, the notion of concurrency and nondeterminism is crucial. When modeling a biological regulatory network, it is necessary to represent the respective evolution of each component of the system. One of the most debated issues with regard to semantics targets the choice of a proper update mode of every component, that is, synchronous [24], asynchronous [52] or more complex ones. The differences and common features of different semantics w.r.t. properties of interest (attractors, oscillators, etc.) have thus resulted in an area of research per itself [19,36,6]. But the biologists often have no idea whether a model of their system of interest should intrinsically be synchronous, asynchronous, generalized... It thus appears crucial to find ways to model systems from raw data without burdening the modelers with an a priori choice of the proper semantics.

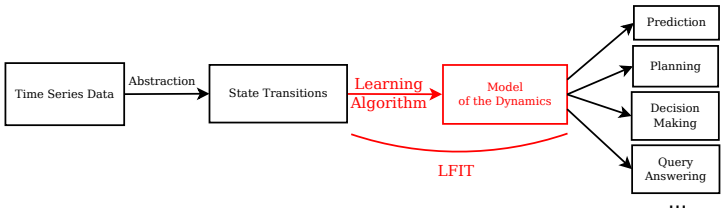


Fig. 1: Assuming a discretization of time series data of a system as state transitions, we propose a method to automatically model the system dynamics.

For a decade, learning dynamics of systems has raised a growing interest in the field of inductive logic programming (ILP) [35,9]. ILP is a form of logic-based machine learning where the goal is to induce a hypothesis (a logic program) that generalises given training examples and background knowledge. Whereas most machine learning approaches learn functions, ILP frameworks learn relations.

In the specific context of learning dynamical systems, previous works proposed an ILP framework entitled learning from interpretation transition (LFIT) [20] 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, several inference algorithms modeling the system as a logic program have been proposed. The semantics of a system's dynamics can indeed differ with regard to the synchronism of its variables, the determinism of its evolution and the influence of its history. The LFIT framework [20,45,43] proposed several modeling and learning algorithms to tackle those different semantics.

In [19,21], state transitions systems are represented with logic programs, in which the state of the world is represented by an Herbrand interpretation and the dynamics that rule the environment changes are represented by a logic program P. The rules in P specify the next state of the world as an Herbrand interpretation through the *immediate consequence operator* (also called the T_P *operator*) [54,2] which mostly corresponds to the synchronous semantics we present in Section 3. In this paper, we extend upon this formalism to model multi-valued variables and any memory-less discrete dynamic semantics including synchronous, asynchronous and general semantics.

[20] proposed the LFIT framework to learn logic programs from traces of interpretation transitions. The learning setting of this framework is as follows. We are given a set of pairs of Herbrand interpretations (I, J) as positive examples such that $J = T_P(I)$, and the goal is to induce a *normal logic program* (NLP) P that realizes the given transition relations. As far as we know, this concept of *learning from interpretation transition* (LFIT) has never been considered in the ILP literature before [20].

To date, the following systems have been tackled: memory-less deterministic systems [20], systems with memory [46], probabilistic systems [32] and their multi-valued extensions [47,31]. [48] proposes a method that allows to deal with continuous time series data, the abstraction itself being learned by the algorithm. As a summary, the systems that LFIT handled so far were restricted to synchronous deterministic dynamics.

In this paper, we extend this framework to learn systems dynamics independently of its update semantics. For this purpose, we propose a modeling of discrete memory-less multi-valued systems as logic programs in which each rule represents that a variable possibly takes some value at the next state, extending the formalism introduced in [20,45]. Research in multi-valued logic programming has proceeded along three different directions [25]: bilattice-based logics [16,18], quantitative rule sets [53] and annotated logics [5,4]. Our representation is based on annotated logics. Here, to each variable corresponds a domain of discrete values. In a rule, a literal is an atom annotated with one of these values. It allows us to represent annotated atoms simply as classical atoms and thus to remain at a propositional level. This modeling allows us 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 propose **GULA**: the General Usage LFIT Algorithm. We show from theoretical results that this algorithm can learn un-

der a wide range of update semantics including synchronous (deterministic or not), asynchronous and generalized semantics.

[43] proposed a first version of **GULA** that we substantially extend in this manuscript. In [43], there was no distinction between feature and target variables, i.e., variables at time step t and $t+1$. From this consideration, interesting properties arise and allow to characterize the kind of semantics compatible with the learning process of the algorithm (Theorem 1). It also allows to represent constraints and to propose a new algorithm (**Synchronizer**, Section 5). We show through theoretical results that this second algorithm can learn a program able to reproduce any given set of discrete state transitions and thus the behavior of any discrete memory-less dynamical semantics.

Empirical evaluation provided in [43] was limited to scalability in complete observability cases. With the goal to proceed real data, we introduce a heuristic method allowing to use **GULA** to learn from partial observations and predict from unobserved data. It allows us to apply the method on more realistic cases by evaluating both scalability, prediction accuracy and explanation of prediction on partial data. Evaluation is performed over the three aforementioned semantics for Boolean network benchmarks from biological literature [27,11]. These experiments emphasize the practical usage of the approach: our implementation reveals to be tractable on systems up to a dozen components, which is sufficient enough to capture a large variety of complex dynamic behaviors in practice.

The organization of the paper is as follows. Section 2 provides a formalization of discrete memory-less dynamics system as multi-valued logic program. Section 3 formalizes dynamical semantics under logic programs. Section 4 presents the first algorithm, **GULA**, which learns optimal programs regardless of the semantics. Section 5 provides extension of the formalization and a second algorithm, the **Synchronizer**, to represent and learn the semantics behavior itself. In Section 6, we propose a heuristic method allowing to use **GULA** to learn from partial observations and predict from unobserved data. Section 7 provides experimental evaluations regarding scalability, prediction accuracy and explanation of predictions. Section 8 discusses related work and Section 9 concludes the paper. All proofs of theorems and propositions are given in Appendix.

2 Logical Modeling of Dynamical Systems

In this section, the concepts necessary to understand the learning algorithms we propose are formalized. In Section 2.1, the basic notions of *multi-valued logic* ($\mathcal{M}$VL) are presented. Then, Section 2.2 presents a modeling of dynamics systems using this formalism. In the following, we denote by $\mathbb{N} := \{0, 1, 2, ...\}$ the set of natural numbers, and for all $k, n \in \mathbb{N}$, $[\![k; n]\!] := \{i \in \mathbb{N} \mid k \leq i \leq n\}$ is the set of natural numbers between k and n included. For any set S, the cardinality of S is denoted $|S|$ and the power set of S is denoted $\wp(S)$.

2.1 Multi-valued Logic Program

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. The atoms of $\mathcal{M}$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 $\mathcal{A}^{\mathcal{V}}_{\mathsf{dom}} = \{\mathrm{v}^{val} \in \mathcal{V} \times \mathcal{V}al \mid val \in \mathsf{dom}(\mathrm{v})\}$ for a given set of variables $\mathcal{V}$ and a given domain function dom. In the following, we work on specific $\mathcal{V}$ and dom that we omit to mention when the context makes no ambiguity, thus simply writing $\mathcal{A}$ for $\mathcal{A}^{\mathcal{V}}_{\mathsf{dom}}$.

Example 1 For a system of 3 variables, the typical set of variables is $\mathcal{V} = \{a, b, c\}$. In general, $\mathcal{V}al = \mathbb{N}$ so that domains are sets of natural integers, for instance: $\mathsf{dom}(a) = \{0, 1\}$, $\mathsf{dom}(b) = \{0, 1, 2\}$ and $\mathsf{dom}(c) = \{0, 1, 2, 3\}$. Thus, the set of all atoms is: $\mathcal{A} = \{a^0, a^1, b^0, b^1, b^2, c^0, c^1, c^2, c^3\}$.

A $\mathcal{M}$VL rule R is defined by:

$$R \quad = \quad \mathrm{v}_0^{val_0} \leftarrow \mathrm{v}_1^{val_1} \wedge \cdots \wedge \mathrm{v}_m^{val_m} \tag{1}$$

where $\forall i \in [\![0; m]\!], \mathrm{v}_i^{val_i} \in \mathcal{A}$ are atoms in $\mathcal{M}$VL so that every variable is mentioned at most once in the right-hand part: $\forall j, k \in [\![1; m]\!], j \neq k \Rightarrow \mathrm{v}_j \neq \mathrm{v}_k$. If $m = 0$, the rule is denoted: $\mathrm{v}_0^{val_0} \leftarrow \top$. Intuitively, 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-hand side of the arrow is called the *head* of R and is denoted $\mathrm{head}(R) := \mathrm{v}_0^{val_0}$. 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 can be 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 it $\emptyset$ if it is empty. The notation $\mathrm{var}(\mathrm{body}(R)) := \{\mathrm{v}_1, \cdots, \mathrm{v}_m\}$ denotes the set of variables that occurs in $\mathrm{body}(R)$. More generally, for all sets of atoms $X \subseteq \mathcal{A}$, we denote $\mathrm{var}(X) := \{\mathrm{v} \in \mathcal{V} \mid \exists val \in \mathsf{dom}(\mathrm{v}), \mathrm{v}^{val} \in X\}$ the set of variables appearing in the atoms of X. A *multi-valued logic program* ($\mathcal{M}$VLP) is a set of $\mathcal{M}$VL rules.

Definition 1 introduces a domination relation between rules that defines a partial anti-symmetric ordering. Intuitively, rules with more general bodies dominate other rules. In our approach, we prefer a more general rule over a more specific one.

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

Example 2 Let $R_1 := a^1 \leftarrow b^1$, $R_2 := a^1 \leftarrow b^1 \wedge c^0$. R_1 dominates R_2 since $\mathrm{head}(R_1) = \mathrm{head}(R_2) = a^1$ and $\mathrm{body}(R_1) \subseteq \mathrm{body}(R_2)$. Intuitively, R_1 is more

general than R_2 on c. R_2 does not dominate R_1 because $\text{body}(R_2) \not\subseteq \text{body}(R_1)$. Let $R_3 := a^1 \leftarrow a^1 \wedge b^0$, R_1 (resp. R_2) does not dominate R_3 (and vice versa), since $\text{body}(R_1) \not\subseteq \text{body}(R_3)$: the rules have a different condition over b. Let $R_4 := a^1 \leftarrow a^1$, for the same reasons, R_1 (resp. R_2) does not dominate R_4. Let $R_5 := a^0 \leftarrow \emptyset$, R_1 (resp. R_2, R_3, R_4) does not dominate R_5 (and vice versa) since their head atoms are different ($a^1 \neq a^0$).

The most general body for a rule is the empty set (also denoted $\top$). A rule with an empty body dominates all rules with the same head atom. Furthermore, the only way two rules dominate each over is that they are the same rule, as stated by Lemma 1.

Lemma 1 (Double Domination Is Equality) *Let R_1, R_2 be two $\mathcal{M}$VL rules. If $R_1 \geq R_2$ and $R_2 \geq R_1$ then $R_1 = R_2$.*

2.2 Dynamic Multi-valued Logic Program

We are interested in modeling non-deterministic (in a broad sense, which includes deterministic) discrete memory-less dynamical systems. In such a system, the next state is decided according to dynamics that depend on the current state of the system. From a modeling perspective, the variables of the system at time step t can be seen as *target variables* and the same variables at time step $t-1$ as *features variables*. Furthermore, additional variables that are external to the system, like *stimuli* or *observation variables* for example, can appear only as feature or target variables. Such a system can be represented by a $\mathcal{M}$VLP with some restrictions. First, the set of variables $\mathcal{V}$ is divided into two disjoint subsets: $\mathcal{T}$ (for targets) encoding system variables at time step t plus optional external variables like observation variables, and $\mathcal{F}$ (for features) encoding system variables at $t-1$ and optional external variables like stimuli. It is thus possible that $|\mathcal{F}| \neq |\mathcal{T}|$. Second, rules only have a conclusion at t and conditions at $t-1$, i.e., only an atom of a variable of $\mathcal{T}$ can be a head and only atoms of variables in $\mathcal{F}$ can appear in a body. In the following, we also re-use the same notations as for the $\mathcal{M}$VL of Section 2.1 such as $\text{head}(R)$, $\text{body}(R)$ and $\text{var}(\text{head}(R))$.

Definition 2 (Dynamic $\mathcal{M}$VLP) Let $\mathcal{T} \subset \mathcal{V}$ and $\mathcal{F} \subset \mathcal{V}$ such that $\mathcal{F} = \mathcal{V} \setminus \mathcal{T}$. A $\mathcal{DM}$VLP P is a $\mathcal{M}$VLP such that $\forall R \in P, \text{var}(\text{head}(R)) \in \mathcal{T}$ and $\forall \text{v}^{val} \in \text{body}(R), \text{v} \in \mathcal{F}$.

In the following, when there is no ambiguity, we suppose that $\mathcal{F}$, $\mathcal{T}$, $\mathcal{V}$ and $\mathcal{A}$ are already defined and we omit to define them again.

Example 3 Figure 2 gives an example of regulation network with three elements a, b and c. The information of this network is not complete; notably, the relative "force" of the components a and b on the component c is not explicit. Multiple dynamics are then possible on this network, among which four possibilities are given below by Program 1 to 4, defined on $\mathcal{T} := \{a_t, b_t, c_t\}$,

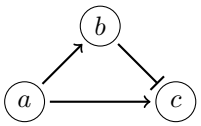


Fig. 2: Example of interaction graph of a regulation network representing an incoherent feed-forward loop [22] where a positively influences b and c, while b (and thus, indirectly, a) negatively influences c.

$\mathcal{F} := \{a_{t-1}, b_{t-1}, c_{t-1}\}$ and $\forall \mathrm{v} \in \mathcal{T} \cup \mathcal{F}, \mathsf{dom}(\mathrm{v}) := \{0, 1\}$. Program 1 is a direct translation of the relations of the regulation network. It only contains rules producing atoms with value 1 which is equivalent to a set of Boolean functions. In Program 2, a always takes value 1 while in Program 3 it always takes value 0, a having no incoming influence in the regulation network this can represent some kind of default behavior. In Program 3, the two red rules introduce potential non-determinism in the dynamics since both conditions can hold at the same time. In Program 4, the rule apply the conditions of the regulation network but it also allows each variable to keep the value 1 at t if it has it at $t-1$ and no inhibition occurs. We insist on the fact that the index notation t or $t-1$ is part of the variable name, not its value. This allows to distinguish variables from $\mathcal{T}$ (t) or $\mathcal{F}$ ($t-1$).

Program 1	**Program 2**	**Program 3**	**Program 4**
$b_t^1 \leftarrow a_{t-1}^1$	$a_t^1 \leftarrow \emptyset$	$a_t^0 \leftarrow \emptyset$	$a_t^1 \leftarrow a_{t-1}^1$
$c_t^1 \leftarrow a_{t-1}^1 \wedge b_{t-1}^0$	$b_t^0 \leftarrow a_{t-1}^0$	$b_t^0 \leftarrow a_{t-1}^0$	$b_t^1 \leftarrow b_{t-1}^1$
	$b_t^1 \leftarrow a_{t-1}^1$	$b_t^1 \leftarrow a_{t-1}^1$	$b_t^1 \leftarrow a_{t-1}^1$
	$c_t^0 \leftarrow a_{t-1}^0$	$c_t^0 \leftarrow a_{t-1}^0$	$c_t^1 \leftarrow c_{t-1}^1 \wedge b_{t-1}^0$
	$c_t^0 \leftarrow b_{t-1}^1$	$c_t^0 \leftarrow b_{t-1}^1$	$c_t^1 \leftarrow a_{t-1}^1 \wedge b_{t-1}^0$
	$c_t^1 \leftarrow a_{t-1}^1 \wedge b_{t-1}^0$	$c_t^1 \leftarrow a_{t-1}^1$	

The dynamical system we want to learn the rules of is represented by a succession of *states* as formally given by Definition 3. We also define the "compatibility" of a rule with a state in Definition 4 and with a transition in Definition 5.

Definition 3 (Discrete state) A *discrete state* s on $\mathcal{T}$ (resp. $\mathcal{F}$) of a $\mathcal{DMVLP}$ is a function from $\mathcal{T}$ (resp. $\mathcal{F}$) to $\mathbb{N}$, i.e., it associates an integer value to each variable in $\mathcal{T}$ (resp. $\mathcal{F}$). It can be equivalently represented by the set of atoms $\{\mathrm{v}^{s(\mathrm{v})} \mid \mathrm{v} \in \mathcal{T}$ (resp. $\mathcal{F})\}$ and thus we can use classical set operations on it. We write $\mathcal{S}^{\mathcal{T}}$ (resp. $\mathcal{S}^{\mathcal{F}}$) to denote the set of all discrete states of $\mathcal{T}$ (resp. $\mathcal{F}$), and a couple of states $(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ is called a *transition*.

When there is no possible ambiguity, we sometimes (Figure 3, Figure 5, ...) denote a state only by the values of variables, without naming the variables. In this case, the variables are given in alphabetical order (a, b, c...). For instance, $\{a^0, b^1\}$ is denoted $\boxed{01}$, $\{a^1, b^0\}$ is denoted $\boxed{10}$ and $\{a^0, b^1, c^0, d^3\}$ is denoted $\boxed{0103}$.

Example 4 Consider a dynamical system having two internal variables a and b, an external stimilus st and an observation variable ch used to trace some important events. The two sets of possible discrete states of a program defined on the two sets of variables $\mathcal{T} = \{a_t, b_t, ch\}$ and $\mathcal{F} = \{a_{t-1}, b_{t-1}, st\}$, and the set of atoms $\mathcal{A} = \{a_t^0, a_t^1, b_t^0, b_t^1, b_t^2, ch^0, ch^1, a_{t-1}^0, a_{t-1}^1, b_{t-1}^0, b_{t-1}^1, b_{t-1}^2, st^0, st^1\}$ are:

$\mathcal{S}^{\mathcal{F}} = \{$
$\{a_{t-1}^0, b_{t-1}^0, st^0\}, \{a_{t-1}^0, b_{t-1}^0, st^1\},$
$\{a_{t-1}^0, b_{t-1}^1, st^0\}, \{a_{t-1}^0, b_{t-1}^1, st^1\},$
$\{a_{t-1}^0, b_{t-1}^2, st^0\}, \{a_{t-1}^0, b_{t-1}^2, st^1\},$
$\{a_{t-1}^1, b_{t-1}^0, st^0\}, \{a_{t-1}^1, b_{t-1}^0, st^1\},$
$\{a_{t-1}^1, b_{t-1}^1, st^0\}, \{a_{t-1}^1, b_{t-1}^1, st^1\},$
$\{a_{t-1}^1, b_{t-1}^2, st^0\}, \{a_{t-1}^1, b_{t-1}^2, st^1\}$ $\}$

and $\mathcal{S}^{\mathcal{T}} = \{$
$\{a_t^0, b_t^0, ch^0\}, \{a_t^0, b_t^0, ch^1\},$
$\{a_t^0, b_t^1, ch^0\}, \{a_t^0, b_t^1, ch^1\},$
$\{a_t^0, b_t^2, ch^0\}, \{a_t^0, b_t^2, ch^1\},$
$\{a_t^1, b_t^0, ch^0\}, \{a_t^1, b_t^0, ch^1\},$
$\{a_t^1, b_t^1, ch^0\}, \{a_t^1, b_t^1, ch^1\},$
$\{a_t^1, b_t^2, ch^0\}, \{a_t^1, b_t^2, ch^1\}$ $\}$.

Here, a_{t-1} and a_t (resp. b_{t-1} and b_t) are theoretically different variables from a $\mathcal{M}$VL perspective. But they actually encode the same variable at different time step and thus a (resp. b) is present in both $\mathcal{F}$ and $\mathcal{T}$ in its corresponding timed form. On the other hand, variables st and ch are respectively a stimuli and an observation variable and thus only appear in $\mathcal{F}, \mathcal{S}^{\mathcal{F}}$ or $\mathcal{T}, \mathcal{S}^{\mathcal{T}}$. Depending on the number of stimuli and observation variables, states of $\mathcal{S}^{\mathcal{F}}$ can have a different size than states in $\mathcal{S}^{\mathcal{T}}$ (see Figure 4).

Definition 4 (Rule-state matching) Let $s \in \mathcal{S}^{\mathcal{F}}$. The $\mathcal{M}$VL rule R *matches* s, written $R \sqcap s$, if $\text{body}(R) \subseteq s$.

We note that this definition of matching only concerns feature variables. Target variables are never meant to be matched.

Example 5 Let $\mathcal{F} = \{a_{t-1}, b_{t-1}, st\}$, $\mathcal{T} = \{a_t, b_t, ch\}$ and $dom(a_{t-1}) = dom(st) = dom(a_t) = dom(ch) = \{0, 1\}, dom(b_{t-1}) = dom(b_t) = \{0, 1, 2\}$. The rule $ch^0 \leftarrow a_{t-1}^1 \wedge b_{t-1}^1 \wedge st^1$ only matches the state $\{a_{t-1}^1, b_{t-1}^1, st^1\}$. The rule $ch^0 \leftarrow a_{t-1}^0 \wedge st^1$ matches $\{a_{t-1}^0, b_{t-1}^0, st^1\}$, $\{a_{t-1}^0, b_{t-1}^1, st^1\}$ and $\{a_{t-1}^0, b_{t-1}^2, st^1\}$. The rule $b_t^2 \leftarrow a_{t-1}^1$ matches $\{a_{t-1}^1, b_{t-1}^0, st^0\}$, $\{a_{t-1}^1, b_{t-1}^0, st^1\}$, $\{a_{t-1}^1, b_{t-1}^1, st^0\}$, $\{a_{t-1}^1, b_{t-1}^1, st^1\}$, $\{a_{t-1}^1, b_{t-1}^2, st^0\}$, $\{a_{t-1}^1, b_{t-1}^2, st^1\}$. The rule $a^1 \leftarrow \emptyset$ matches all states of $\mathcal{S}^{\mathcal{F}}$.

The final program we want to learn should both:

- match the observations in a complete (all transitions are learned) and correct (no spurious transition) way;
- represent only minimal necessary interactions (according to Occam's razor: no overly-complex bodies of rules)

The following definitions formalize these desired properties. In Definition 5 we characterize the fact that a rule of a program is useful to describe the dynamics of one variable in a transition; this notion is then extended to a program and a set of transitions, under the condition that there exists such a rule for each variable and each transition. A conflict (Definition 6) arises when a rule describes a change that is not featured in the considered set of transitions. Finally, Definition 8 and Definition 7 give the characteristics of

a complete (the whole dynamics is covered) and consistent (without conflict) program.

Definition 5 (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'), written $s \xrightarrow{R} s'$, if $R \sqcap s \wedge \text{head}(R) \in s'$.

A $\mathcal{DM}$VLP P *realizes* $(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, written $s \xrightarrow{P} s'$, if $\forall \text{v} \in \mathcal{T}, \exists R \in P, \text{var}(\text{head}(R)) = \text{v} \wedge s \xrightarrow{R} s'$. It *realizes* a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, written $\overset{P}{\hookrightarrow} T$, if $\forall (s, s') \in T, s \xrightarrow{P} s'$.

Example 6 The rule $c_t^1 \leftarrow a_{t-1}^1 \wedge b_{t-1}^1$ realizes the transition $t = (\{a_{t-1}^1, b_{t-1}^1, c_{t-1}^0\}, \{a_t^0, b_t^1, c_t^1\})$ since it matches the first state of t and its conclusion is in the second state. However, the rule $c_t^1 \leftarrow a_{t-1}^1 \wedge b_{t-1}^0$ does not realize t since it does not match the feature state of t.

Example 7 The transition $t = (\{a_{t-1}^1, b_{t-1}^1, c_{t-1}^0\}, \{a_t^0, b_t^1, c_t^1\})$ is realized by Program 3 of Example 3, by using the rules $a_t^0 \leftarrow \emptyset$, $b_t^1 \leftarrow a_{t-1}^1$ and $c_t^1 \leftarrow a_{t-1}^1$. However, Program 2 of the same Example does not realize t since the only rule that could produce c_t^1, that is, $c_t^1 \leftarrow a_{t-1}^1 \wedge b_{t-1}^0$, does not match $\{a_{t-1}^1, b_{t-1}^1, c_{t-1}^0\}$; moreover, no rule can produce a_t^0. Programs 1 and 4 of the same Example cannot produce a_t^0 either and thus do not realize t.

In the following, for all sets of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, we denote: $\text{first}(T) := \{s \in \mathcal{S}^{\mathcal{F}} \mid \exists (s_1, s_2) \in T, s_1 = s\}$ the set of all initial states of these transitions. We note that $\text{first}(T) = \emptyset \iff T = \emptyset$.

Definition 6 (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 $\exists s \in \text{first}(T), \big(R \sqcap s \wedge \forall (s, s') \in T, \text{head}(R) \notin s'\big)$. R is said to be *consistent* with T when R does not conflict with T.

A rule is consistent if for all initial states of the transitions of T (first(T)) matched by the rule, there exists a transitions of T for which it verifies the conclusion.

Definition 7 (Consistent program) A $\mathcal{DM}$VLP P is *consistent* with a set of transitions T if P does not contain any rule R conflicting with T.

Example 8 Let $s1 = \{a_{t-1}^1, b_{t-1}^0, c_{t-1}^0\}, s2 = \{a_{t-1}^1, b_{t-1}^0, c_{t-1}^1\}, s3 = \{a_{t-1}^0, b_{t-1}^0, c_{t-1}^0\}$ and $t1 = (s1, \{a_t^0, b_t^1, c_t^1\})$,
$t2 = (s1, \{a_t^1, b_t^1, c_t^0\})$,
$t3 = (s2, \{a_t^0, b_t^1, c_t^0\})$,
$t4 = (s2, \{a_t^0, b_t^0, c_t^1\})$,
$t5 = (s3, \{a_t^1, b_t^1, c_t^0\})$.
Let $T = \{t1, t2, t3, t4, t5\}$.

Program 1 of Example 3 is consistent with T. The rule $b_t^1 \leftarrow a_{t-1}^1$ matches $s1$ and both $s1$ and b_t^1 are observed in $t2$. The rule also matches $s2$ which is

observed with b_t^1 in $t3$. The rule $c_t^1 \leftarrow a_{t-1}^1 \wedge b_{t-1}^0$ matches $s1$ (resp. $s2$), which is observed with c_t^1 in $t1$ (resp. $t3$).

Program 2 is not consistent with T since $a_t^1 \leftarrow \emptyset$ is not consistent with T: it matches $s1$, $s2$ and $s3$ but the transitions of T that include $s2$ ($t3$, $t4$) do not contain a_t^1. Program 3 is not consistent with T since $a_t^0 \leftarrow \emptyset$ matches $s1$, $s2$, $s3$ but the only transition that contains $s3$ ($t5$) does not contain a_t^0. Program 4 is not consistent with T since $a_t^1 \leftarrow a_{t-1}^1$ matches $s2$ but the transitions of T that include $s2$ ($t3$, $t4$) do not contain a_t^1.

Definition 8 (Complete program) A $\mathcal{DM}$VLP P is *complete* if $\forall s \in \mathcal{S}^{\mathcal{F}}, \forall \mathrm{v} \in \mathcal{T}, \exists R \in P, R \sqcap s \wedge \mathrm{var}(\mathrm{head}(R)) = \mathrm{v}$.

A complete $\mathcal{DM}$VLP realizes at least one transition for each possible initial state.

Example 9 Program 1 of Example 3 is not complete since it does not have any rule over target variable a_t, in fact it does not realize any transitions. Program 2 of same example is complete:

- The rule $a_t^1 \leftarrow \emptyset$ will realize a_t^1 from any feature state;
- For b_t it has a first (resp. second) rule that matches all feature state where a_{t-1}^0 (resp. a_{t-1}^1) appears and the domain of a_{t-1} being $\{0, 1\}$ all cases and thus all feature states are covered by this two rules;
- For c_t, all combinations of values of a and b are covered by the three last rules, $\forall val \in \mathsf{dom}(c_{t-1})$,
  - $\{a_{t-1}^0, b_{t-1}^0, c_{t-1}^{val}\}$ is matched by $c_t^0 \leftarrow a_{t-1}^0$;
  - $\{a_{t-1}^0, b_{t-1}^1, c_{t-1}^{val}\}$ is matched by $c_t^0 \leftarrow b_{t-1}^1$ (and $c_t^0 \leftarrow b_{t-1}^1$);
  - $\{a_{t-1}^1, b_{t-1}^0, c_{t-1}^{val}\}$ is matched by $c_t^0 \leftarrow a_{t-1}^1 \wedge b_{t-1}^0$;
  - $\{a_{t-1}^1, b_{t-1}^1, c_{t-1}^{val}\}$ is matched by $c_t^0 \leftarrow b_{t-1}^1$.

Program 3 is also complete, and it even realizes multiple values for c_t when both a_{t-1}^1 and b_{t-1}^1 are in a feature state: $\{a_{t-1}^1, b_{t-1}^1, c_{t-1}^0\}$ is matched by both $c_t^0 \leftarrow b_{t-1}^1$ and $c_t^1 \leftarrow a_{t-1}^1$. Program 4 is not complete: no transition is realized when a_{t-1}^0 is in a feature state since the only rule of a_t is $a_t^1 \leftarrow a_{t-1}^1$.

Definition 9 groups all the properties that we want the learned program to have: suitability and optimality, and Proposition 1 states that the optimal program of a set of transitions is unique.

Definition 9 (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 when:

- P is consistent with T,
- P realizes T,
- P is complete,
- for any possible $\mathcal{M}$VL rule R consistent with T, there exists $R' \in P$ such that $R' \geq R$.

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*.

Note that Definition 9 ensures local minimality. In terms of biological models, it is more interesting to focus on local minimality, thus simple but numerous rules, modeling local influences from which the complexity of the whole system arises, than global minimality that would produce system-level rules hiding the local correlations and influences. Definition 9 also guarantees that we obtain all the minimal rules which guarantees to provide biological collaborators with the whole set of possible explanations of biological phenomena involved in the system of interest.

Proposition 1 (Uniqueness of Optimal Program) *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. The $\mathcal{DM}$VLP optimal for T is unique and denoted $P_{\mathcal{O}}(T)$.*

Example 10

$$\begin{aligned}\text{Let } T = \{\quad & (\{a^0_{t-1}, b^0_{t-1}, c^0_{t-1}\}, \{a^1_t, b^0_t, c^0_t\})\\ & (\{a^0_{t-1}, b^0_{t-1}, c^1_{t-1}\}, \{a^1_t, b^0_t, c^0_t\})\\ & (\{a^0_{t-1}, b^1_{t-1}, c^0_{t-1}\}, \{a^1_t, b^0_t, c^0_t\})\\ & (\{a^1_{t-1}, b^0_{t-1}, c^0_{t-1}\}, \{a^1_t, b^1_t, c^1_t\})\\ & (\{a^0_{t-1}, b^1_{t-1}, c^1_{t-1}\}, \{a^1_t, b^0_t, c^0_t\})\\ & (\{a^1_{t-1}, b^0_{t-1}, c^1_{t-1}\}, \{a^1_t, b^1_t, c^1_t\})\\ & (\{a^1_{t-1}, b^1_{t-1}, c^0_{t-1}\}, \{a^1_t, b^1_t, c^0_t\})\\ & (\{a^1_{t-1}, b^1_{t-1}, c^0_{t-1}\}, \{a^1_t, b^1_t, c^1_t\})\quad \}\ .\end{aligned}$$

Program 1 and 4 of Example 3 are not complete (see Example 9) and thus not suitable for T. Program 3 is complete but not consistent with T (see Example 8). Program 2 is complete, consistent and realizes T but is not suitable for T: indeed, $c^1_t \leftarrow a^1_{t-1}$ is consistent with T and there is no rule in Program 2 that dominates it.

Let us consider:

$$\begin{aligned}P := \{\quad & a^1_t \leftarrow \emptyset\\ & b^0_t \leftarrow a^0_{t-1}\\ & b^1_t \leftarrow a^1_{t-1}\\ & c^0_t \leftarrow a^0_{t-1}\\ & c^0_t \leftarrow b^1_{t-1}\\ & c^1_t \leftarrow a^1_{t-1}\\ & c^1_t \leftarrow a^1_{t-1} \wedge b^0_{t-1}\quad \}\ .\end{aligned}$$

P is complete, consistent, realizes T and all rules consistent with T are dominated by a rule of P. Thus, P is suitable for T. But P is not optimal since $c^1_t \leftarrow a^1_{t-1} \wedge b^0_{t-1}$ is dominated by $c^1_t \leftarrow a^1_{t-1}$. By removing $c^1_t \leftarrow a^1_{t-1} \wedge b^0_{t-1}$ from P, we obtain the optimal program of T.

Algorithm 1 **Brute Force Enumeration**

- **INPUT:** a set of atoms $\mathcal{A}$, two sets of variables $\mathcal{F}$ and $\mathcal{T}$ and a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$.
- Generate all possible rules over $\mathcal{A}, \mathcal{F}, \mathcal{T}$.
  - $P := \{\mathrm{v}^{val} \leftarrow \{\mathrm{v}'^{val'} \mid \mathrm{v}'^{val'} \in \mathcal{A} \wedge \mathrm{v}' \in \mathcal{F}\} \mid \mathrm{v}^{val} \in \mathcal{A} \wedge \mathrm{v} \in \mathcal{T}\}$
- Keep only the rules consistent with T.
  - $P := \{R \in P \mid \forall (s, s') \in T, \mathrm{body}(R) \subseteq s \implies \exists (s, s'') \in T, \mathrm{head}(R) \in s''\}$
- Remove rules dominated by another rule
  - $P := \{R \in P \mid \nexists R' \in P, R' \neq R \wedge R' \geq R\}$
- **OUTPUT:** P (P is $P_{\mathcal{O}}(T)$).

According to Definition 9, we can obtain the optimal program by a trivial brute force enumeration: generate all rules consistent with T then remove the dominated ones as shown in Algorithm 1.

The purpose of Section 4 is to propose a non-trivial approach that is more efficient in practice to obtain the optimal program. This approach also respects the optimality properties of Definition 9 and thus ensures independence from the dynamical semantics, that are detailed in next Section.

3 Dynamical semantics

The aim of this section is to formalize the general notion of *dynamical semantics* as an update policy based on a program, and to give characterizations of several widespread existing semantics used on discrete models.

In the previous section, we supposed the existence of two distinct sets of variables $\mathcal{F}$ and $\mathcal{T}$ that represent conditions (features) and conclusions (targets) of rules. Conclusion atoms allow to create one or several new state(s) made of target variables, from conditions on the current state which is made of feature atoms.

In Definition 10, we formalize the notion of dynamical semantics which is a function that, to a program, associates a set of transitions where each state has at least one outgoing transition. Such a set of transitions can also be seen as a function that maps any state to a non-empty set of states, regarded as possible dynamical branchings. We give examples of semantics afterwards.

Definition 10 (Dynamical Semantics) A *dynamical semantics* (on $\mathcal{A}$) is a function that associates, to each $\mathcal{DM}$VLP P, a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ so that: $\mathrm{first}(T) = \mathcal{S}^{\mathcal{F}}$. Equivalently, a dynamical semantics can be seen as a function of $\left(\mathcal{DM}\text{VLP} \rightarrow (\mathcal{S}^{\mathcal{F}} \rightarrow \wp(\mathcal{S}^{\mathcal{T}}) \setminus \{\emptyset\})\right)$ where $\mathcal{DM}$VLP is the set of $\mathcal{DM}$VLPs.

A dynamical semantics has an infinity of possibility to produce transitions from a $\mathcal{DM}$VLP. Indeed, like $DS_1(P)$ of Example 11, a semantics can totally ignore the $\mathcal{DM}$VLP rules. It can also use the rule in an adversary way

like $DS_{inverse}$ that keeps only the transitions that are not permitted by the program. Such semantics can produce transitions that are not consistent with the input program, i.e., the rules which conclusions were not selected for the transition will be in conflict with the set of transitions from this feature state. The kind of semantics we are interested in are the ones that properly use the rule of the $\mathcal{DM}$VLP and ensure the properties of consistency introduced in Definition 7.

In Example 11, the dynamical semantics DS_{syn}, DS_{asyn} and DS_{gen} are example of such semantics. They are trivial forms of the synchronous, asynchronous and general semantics that are widely used in bioinformatics. Indeed, DS_{syn} is trivial because it generates transitions towards an arbitrary state when the program P is not complete (if no rule matches for some target variable, the program produces an incomplete state), while DS_{asyn} and DS_{gen} are trivial because they require feature and target variables to correspond and have a specific form (labelled with $t-1$ and t) with no additional stimuli or observation variables. We formalize those three semantics properly under our modeling in next Section with no restriction on the feature and target variables forms.

Example 11 For this example, suppose that feature and target variable are "symmetrical" (called *regular variables* later): $\mathcal{T} = \{a_t, b_t, ..., z_t\}$ and $\mathcal{F} = \{a_{t-1}, b_{t-1}, ..., z_{t-1}\}$, with: $\forall x_t, x_{t-1} \in \mathcal{T} \times \mathcal{F}, \mathsf{dom}(x_T) = \mathsf{dom}(x_{t-1})$. Let *convert* be a function of $(\mathcal{S}^{\mathcal{F}} \to \mathcal{S}^{\mathcal{T}})$ such that for any $\mathcal{DM}$VLP $P, \forall s \in \mathcal{S}^{\mathcal{F}}, convert(s) = \{\mathrm{v}_t^{val} \mid \mathrm{v}_{t-1}^{val} \in s\}$, and $s_0 \in \mathcal{S}^{\mathcal{T}}$ an arbitrary target state that is used to ensure that each of the following semantics produces at least one target state. Let DS_1, DS_2, DS_{syn}, DS_{asyn}, DS_{gen} and $DS_{inverse}$ be dynamical semantics defined as follows, where P is a $\mathcal{DM}$VLP and $s \in \mathcal{S}^{\mathcal{F}}$:

- $(DS_1(P))(s) = \{s_0\}$
- $(DS_2(P))(s) = \{s' \in \mathcal{S}^{\mathcal{T}} \mid s' \in \{\mathrm{head}(R) \mid R \in P, |\,\mathrm{body}(R)| = 3\}\} \cup \{s_0\}$
- $(DS_{syn}(P))(s) = \{s' \in \mathcal{S}^{\mathcal{T}} \mid s' \in \{\mathrm{head}(R) \mid R \in P, \mathrm{body}(R) \subseteq s\}\} \cup \{s_0\}$
- $(DS_{asyn}(P))(s) = \{s' \in \mathcal{S}^{\mathcal{T}} \mid s' \in convert(s) \cup \{\mathrm{head}(R) \mid R \in P, \mathrm{body}(R) \subseteq s\} \wedge |\{\mathrm{v}_t^{val} \in s' \mid \mathrm{v}_{t-1}^{val} \in s\}| \in \{|\mathcal{T}|, |\mathcal{T}| - 1\}\}$
- $(DS_{gen}(P))(s) = \{s' \in \mathcal{S}^{\mathcal{T}} \mid s' \in convert(s) \cup \{\mathrm{head}(R) \mid R \in P, \mathrm{body}(R) \subseteq s\}\}$
- $(DS_{inverse}(P))(s) = (\mathcal{S}^{\mathcal{T}} \setminus (DS_{syn}(P))(s)) \cup \{s_0\}$

DS_1 always outputs transitions towards s_0 and totally ignores the rules of the given program and thus can produce transitions that are not consistent with the input program. DS_2 uses the rules of the $\mathcal{DM}$VLP but in an improper way, as it always considers the conclusions of rules as long as they have exactly 3 conditions, whether they match the feature state or not. $DS_{inverse}$ uses proper rules conclusions, but in order to contradict the program: it produces transitions so that the program is not consistent, plus a transition to s_0 to ensure at least a transition.

DS_{syn} use the rules in the expected way, i.e., it checks if they match the considered feature state and applies their conclusion; it is a trivial form of

synchronous semantics as properly introduced later in Definition 14. DS_{asyn} also uses the rules as expected: it uses the feature state to restrict the possible target states to at most one modification compared to the feature state; this is a trivial form of asynchronous semantics, as properly introduced later in Definition 15. DS_{gen} also uses the rules as expected: it mixes the current feature state with rules conclusions to produce a partially new target state; it is a trivial form of general semantics, as properly introduced later in Definition 16.

We now aim at characterizing a set of semantics of interest for the current work, as given in Theorem 1. Beforehand, Definition 11 allows to denote as $\mathsf{Conclusions}(s, P)$ the set of heads of rules, in a program P, matching a state s, and Definition 12 introduces a notation $B|_X$ to consider only atoms in a set $B \subseteq \mathcal{A}$ that have their variable in a set $X \subseteq \mathcal{V}$. These two notations will be used in the next theorem and afterwards. In the following, we especially use the notation of Definition 12 with $\mathcal{A}$ (denoted $\mathcal{A}|_X$) and on $\mathsf{Conclusions}$ (denoted $\mathsf{Conclusions}|_X(s, P)$).

Definition 11 (Program Conclusions) Let s in $\mathcal{S}^{\mathcal{F}}$ and P a $\mathcal{M}$VLP. We denote: $\mathsf{Conclusions}(s, P) := \{\text{head}(R) \in \mathcal{A} \mid R \in P, R \sqcap s\}$ the set of conclusion atoms in state s for the program P.

Definition 12 (Restriction of a Set of Atoms) Let $B \subseteq \mathcal{A}$ be a set of atoms, and $X \subseteq \mathcal{V}$ be a set of variables. We denote: $B|_X = \{\mathrm{v}^{val} \in B \mid \mathrm{v} \in X\}$ the set of atoms of B that have their variables in X. If B is instead a function that outputs a set of atoms, we note $B|_X(params)$ instead of $\big(B(params)\big)|_X$, where $params$ is the sequence of parameters of B.

With Theorem 1, we characterize semantics which for any $\mathcal{DM}$VLP produce the same behavior using the corresponding optimal program, that is, any semantics DS such that for any $\mathcal{DM}$VLP $P, DS(P) = DS(P_{\mathcal{O}}(DS(P)))$. Such a semantics produces new states based only on the initial state s and the heads of matching rules of the given program $\mathsf{Conclusions}(s, P)$, as stated by point (2). Moreover, $P_{\mathcal{O}}(DS(P))$ being consistent with $DS(P)$, given a state $s \in \mathcal{S}^{\mathcal{F}}$, $\mathsf{Conclusions}(s, P_{\mathcal{O}}(DS(P))) = \bigcup_{s' \in DS(P)(s)} s'$, i.e., all the target atoms observed in a target state of $DS(P)(s)$ will be the head of some rule that matches s in the optimal program. In other words, it will be given to the semantics to choose from when the program $P_{\mathcal{O}}(DS(P))$ is used with semantics DS. Thus the semantics should produce the same states, when being given the atoms of all those next states as possibilities, as stated by point (1). Those two conditions are sufficient to ensure that $DS(P_{\mathcal{O}}(DS(P))) = DS(P)$ and thus can be used to assert if the dynamics of a given semantics, for any given original program P, can be reproduced using the corresponding optimal program $P_{\mathcal{O}}(DS(P))$ with the same semantics.

Theorem 1 (Pseudo-idempotent Semantics and Optimal $\mathcal{DM}$VLP) *Let DS be a dynamical semantics.*
If, for all P a $\mathcal{DM}$VLP, there exists $\mathsf{pick} \in (\mathcal{S}^{\mathcal{F}} \times \wp(\mathcal{A}|_{\mathcal{T}}) \to \wp(\mathcal{S}^{\mathcal{T}}) \setminus \{\emptyset\})$ *so that:*

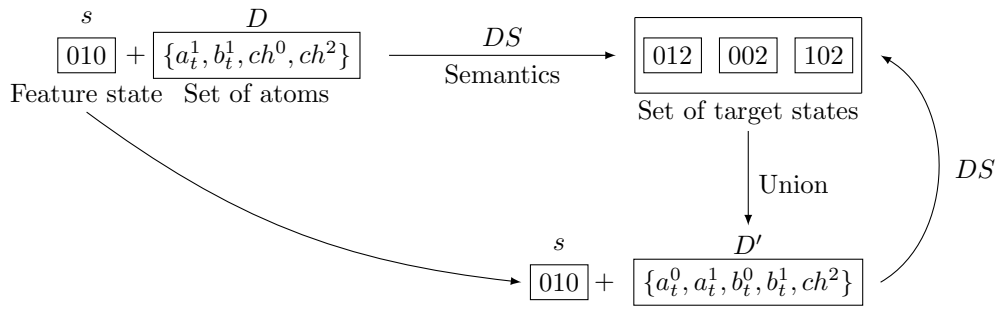


Fig. 3: Example of a pseudo-idempotent semantics DS.

(1) $\forall D \subseteq \mathcal{A}|_{\mathcal{T}}, \mathsf{pick}(s, \bigcup_{s' \in \mathsf{pick}(s,D)} s') = \mathsf{pick}(s, D)$, *and*
(2) $\forall s \in \mathcal{S}^{\mathcal{F}}, \big(DS(P)\big)(s) = \mathsf{pick}(s, \mathsf{Conclusions}(s, P))$,

then, for all P *a* $\mathcal{DM}$VLP, $DS(P_{\mathcal{O}}(DS(P)))) = DS(P)$.

Example 12 Let DS be a dynamical semantics, $s \in \mathcal{S}^{\mathcal{F}}$ be a feature state such that $s = \{a_{t-1}^0, b_{t-1}^1, st^0\}$, P be a $\mathcal{DM}$VLP such that $\mathsf{Conclusions}(P, s) = \{a_t^1, b_t^1, ch^0, ch^2\}$. In Figure 3, from s and $\mathsf{Conclusions}(P, s)$, DS produces three different target states, i.e., $(DS(P))(s) = \mathsf{pick}(s, \mathsf{Conclusions}(s, P)) = \{\{a_t^0, b_t^1, ch^2\}, \{a_t^0, b_t^0, ch^2\}, \{a_t^1, b_t^0, ch^2\}\}$. Let $D = \mathsf{Conclusions}(P, s)$, here, the set of occurring atoms in the states produced by $DS(s, D)$ is $D' = \bigcup_{s' \in \mathsf{pick}(s,D)} =$ $\{\mathbf{a_t^0}, a_t^1, \mathbf{b_t^0}, b_t^1, ch^2\}$. In this example, the function pick uses all target atoms of D except ch^0 and introduces two additional atoms $\mathbf{a_t^0}$, $\mathbf{b_t^0}$, it also only produces 3 of the 4 possible target states composed of those atoms: this semantics does not allows a_t^1 and b_t^1 to appear together in transition from s. If we call the function pick by replacing the program conclusions by the semantics conclusions we observe the same resulting states, i.e., $\mathsf{pick}(s, D') = \mathsf{pick}(s, D)$. Given the target atoms selected by the semantics, it reproduces the same set of target states in this example; if the semantics has this behavior for any feature state s and any program P, it is pseudo-idempotent.

Up to this point, no link has been made between corresponding feature (in $\mathcal{F}$) and target (in $\mathcal{T}$) variables or atoms. In other words, the formal link between the two atoms v_t^{val} and v_{t-1}^{val} with the same value has not been made yet. This link, called *projection*, is established in Definition 13, under the only assumption that $\mathsf{dom}(\mathrm{v}_t) = \mathsf{dom}(\mathrm{v}_{t-1})$. It has two purposes:

– When provided with a set of transitions, for instance by using a dynamical semantics, one can describe dynamical paths, that is, successions of next states, by using each next state to generate the equivalent initial state for the next transition;

– Some dynamical semantics (such as the asynchronous one, see Definition 15) make use of the current state to build the next state, and as such need a way to convert target variables into feature variables.

However, such a projection cannot be defined on the whole sets of target ($\mathcal{T}$) and feature ($\mathcal{F}$) variables, but only on two subsets $\overline{\mathcal{F}} \subseteq \mathcal{F}$ and $\overline{\mathcal{T}} \subseteq \mathcal{T}$. Note that we require the projection to be a bijection, thus: $|\overline{\mathcal{F}}| = |\overline{\mathcal{T}}|$. These subsets $\overline{\mathcal{T}}$ and $\overline{\mathcal{F}}$ contain variables that we call afterwards *regular variables*: they correspond to variables that have an equivalent in both the initial states (at $t-1$) and the next states (at t). Variables in $\mathcal{F} \setminus \overline{\mathcal{F}}$ can be considered as *stimuli* variables: they can only be observed in the previous state but we do not try to explain their next value in the current state; this is typically the case of external stimuli (sun, stress, nutriment...) that are unpredictable when observing only the studied system. Variables in $\mathcal{T} \setminus \overline{\mathcal{T}}$ can be considered as *observation* variables: they are only observed in the present state as the result of the combination of other (regular and stimuli) variables; they can be of use to assess the occurrence of a specific configuration in the previous state but cannot be used to generate the next step. For the rest of this section, we suppose that $\overline{\mathcal{F}}$ and $\overline{\mathcal{T}}$ are given and that there exists such projection functions, as given by Definition 13. Figure 4 gives a representation of these sets of variables.

It is noteworthy that projections on states are not bijective, because of stimuli variables that have no equivalent in target variables, and observation variables that have no equivalent in feature variables (see Figure 4). Therefore, the focus is often made on regular variables (in $\overline{\mathcal{F}}$ and $\overline{\mathcal{T}}$). Especially, for any pair of states $(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, having $\mathsf{sp}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}}(s') \subseteq s$, which is equivalent to $\mathsf{sp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(s) \subseteq s'$, means that the regular variables in s and their projection in s' (or conversely) hold the same value, modulo the projection.

Definition 13 (Projections) A *projection on variables* is a bijective function $\mathsf{vp}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}} : \overline{\mathcal{T}} \to \overline{\mathcal{F}}$ so that $\overline{\mathcal{T}} \subseteq \mathcal{T}$, $\overline{\mathcal{F}} \subseteq \mathcal{F}$, and: $\forall \mathrm{v} \in \overline{\mathcal{T}}, \mathsf{dom}(\mathsf{vp}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}}(\mathrm{v})) = \mathsf{dom}(\mathrm{v})$. The *projection on atoms* (based on $\mathsf{vp}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}}$) is the bijective function:

$$\begin{aligned} \mathsf{ap}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}} : \mathcal{A}|_{\overline{\mathcal{T}}} &\to \mathcal{A}|_{\overline{\mathcal{F}}} \\ \mathrm{v}^{val} &\mapsto \left(\mathsf{vp}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}}(\mathrm{v})\right)^{val} \quad . \end{aligned}$$

The inverse function of $\mathsf{vp}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}}$ is denoted $\mathsf{vp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}$ and the inverse function of $\mathsf{ap}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}}$ is denoted $\mathsf{ap}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}$.

The *projections on states* (based on $\mathsf{ap}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}}$ and $\mathsf{ap}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}$) are the functions:

$$\begin{aligned} \mathsf{sp}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}} : \mathcal{S}^{\mathcal{T}} &\to \mathcal{S}^{\overline{\mathcal{F}}} \\ s' &\mapsto \{\mathsf{ap}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}}(\mathrm{v}^{val}) \in \mathcal{A} \mid \mathrm{v}^{val} \in s' \wedge \mathrm{v} \in \overline{\mathcal{T}}\} \\ \mathsf{sp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}} : \mathcal{S}^{\mathcal{F}} &\to \mathcal{S}^{\overline{\mathcal{T}}} \\ s &\mapsto \{\mathsf{ap}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(\mathrm{v}^{val}) \in \mathcal{A} \mid \mathrm{v}^{val} \in s \wedge \mathrm{v} \in \overline{\mathcal{F}}\} \quad . \end{aligned}$$

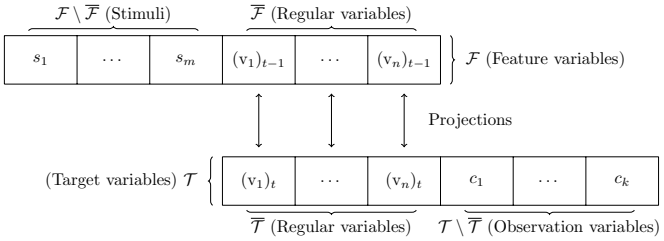


Fig. 4: Representation of a state transition of a dynamical system over n variables, m stimuli and k observation variables, i.e., $|\mathcal{F}| = n+m, |\mathcal{T}| = n+k$.

Example 13 In Example 12, there are three feature variables (a_{t-1}, b_{t-1}, st) and three target variables (a_t, b_t, ch). If we consider that the regular variables are $\overline{\mathcal{T}} = \{a_t, b_t\}$ and $\overline{\mathcal{F}} = \{a_{t-1}, b_{t-1}\}$, we can define the following (bijective) projection on variables: $\mathsf{vp}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}} : \begin{cases} a_t \mapsto a_{t-1} \\ b_t \mapsto b_{t-1} \end{cases}$. Following Definition 13, we have, for instance:

- $\mathsf{ap}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}}(a_t^1) = a_{t-1}^1$,
- $\mathsf{ap}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(b_{t-1}^0) = b_t^0$,
- $\mathsf{sp}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}}(\{a_t^0, b_t^0, ch^0\}) = \{a_{t-1}^0, b_{t-1}^0\}$, and
- $\mathsf{sp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(\{a_{t-1}^1, b_{t-1}^0, st^1\}) = \{a_t^1, b_t^0\}$.

3.1 Synchronous, Asynchronous and General Semantics

In the following, we present a formal definition and a characterization of three particular semantics that are widespread in the field of complex dynamical systems: synchronous, asynchronous and general.Note that some points in these definitions are arbitrary and could be discussed depending on the modeling paradigm. For instance, the policy about rules R so that $\exists s \in \mathcal{S}^{\mathcal{F}}, R \sqcap s \wedge \mathsf{ap}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}}(\mathrm{head}(R)) \in s$, which model stability in the dynamics, could be to include them (such as in the synchronous and general semantics) or exclude them (such as in the asynchronous semantics) from the possible dynamics. The modeling method presented so far in this paper is independent to the considered semantics as long as it respects Definition 10 and the capacity of the optimal program to reproduce the observed behavior is ensured as long as the semantics respects Theorem 1.

Definition 14 introduces the synchronous semantics, consisting in updating all variables at once in each step in order to compute the next state. The value of each variable in the next state is taken amongst a "pool" of atoms containing all conclusions of rules that match the current state (using Conclusions) and atoms produced by a "default function" d that is explained below. However, this is taken in a loose sense: as stated above, atoms that make a variable

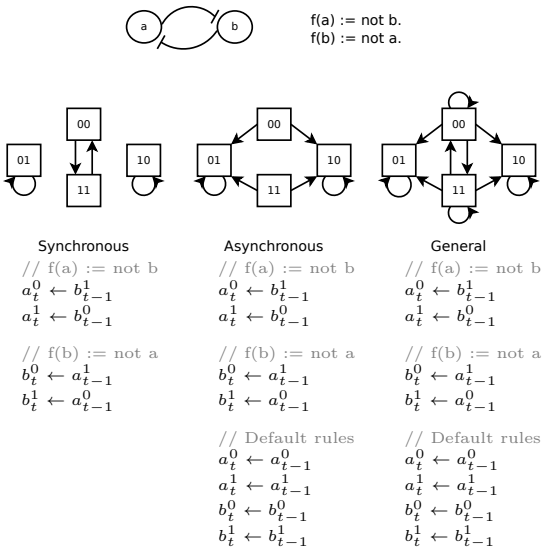


Fig. 5: A Boolean network with two variables inhibiting each other (top). The corresponding synchronous, asynchronous and general dynamics are given as state-transition diagrams (middle). In these state-transition diagrams, each box with a label "xy" represents both the feature state $\{a_{t-1}^x, b_{t-1}^y\}$ and the target state $\{a_t^x, b_t^y\}$, and each arrow represents a possible transitions between states. The corresponding optimal $\mathcal{DM}$VLP (bottom) contain comments (in grey) that explain sub-parts of the programs.

change its value are not prioritized over atoms that don't. Furthermore, if several atoms on the same variable are provided in the pool (as conclusions of different rules or provided by the default function), then several transitions are possible, depending on which one is chosen. Thus, for a self-transition $(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ with $\mathsf{sp}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}}(s') \subseteq s$ to occur, there needs to be, for each atom $\mathrm{v}^{val} \in s'$ so that $\mathrm{v} \in \overline{\mathcal{T}}$, either a rule that matches s and whose head is v^{val} or that the default function gives the value v^{val}. Note however that such a loop is not necessarily a point attractor (that is, a state for which the only possible transition is the self-transition); it is only the case if all atoms in the pool are also in $\mathsf{sp}_{\overline{\mathcal{T}} \to \overline{\mathcal{F}}}(s)$.

As explained above, for a given state s and a given set of variables W, the function d provides a set of "default atoms" added to the pool of atoms used to build the next state, along with rules conclusions. This function d, however, is not explicitly given: the only constraints are that:

- d produces atoms at least for a provided set of variables W, specifically, the set of variables having no conclusion in a given state, which is necessary in the case of an incomplete program,
- $d(s, \emptyset)$ is a subset of $d(s, W)$ for all W, as it intuitively represents a set of default atoms that are always available.

Note that $d(s, \emptyset) = \emptyset$ always respects these constraints and is thus always a possible value. In the case of a complete program, that is, a program providing conclusions for every variables in every state, d is always called with $W = \emptyset$ and the other cases can thus be ignored. Another typical use for d is the case of a system with Boolean variables (i.e., such that $\forall \mathrm{v} \in \mathcal{V}, \mathsf{dom}(\mathrm{v}) = \{0, 1\}$) where a program P is built by importing only the positive rules of the system, that is, only rules with atoms v_t^1 as heads. This may happen when importing a model from another formalism featuring only Boolean formulas, such as Boolean networks. In this case, d can be used to provide a default atom w_t^0 for all variables w that do not appear in $\mathsf{Conclusions}(s, P)$, thus reproducing the dynamics of the original system.

Definition 14 (Synchronous semantics) Let $d \in (\mathcal{S}^{\mathcal{F}} \times \wp(\mathcal{T}) \to \wp(\mathcal{A}|_{\mathcal{T}}))$, so that $\forall s \in \mathcal{S}^{\mathcal{F}}, \forall W \subseteq \mathcal{T}, W \subseteq \mathrm{var}(d(s, W)) \wedge d(s, \emptyset) \subseteq d(s, W)$. The *synchronous semantics* $\mathcal{T}_{syn}$ is defined by:

$$\mathcal{T}_{syn} : P \mapsto \{(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}} \mid s' \subseteq \mathsf{Conclusions}(s, P) \cup d(s, \mathcal{T} \setminus \mathrm{var}(\mathsf{Conclusions}(s, P)))\}$$

Example 14 It is possible to reproduce classical Boolean network dynamics using the synchronous semantics ($\mathcal{T}_{syn}$) with a well-chosen default function. Indeed, Boolean models are classically defined as a set of Boolean function providing conditions in which each variable becomes active, thus implying that all the other cases make them inactive. A straightforward translation of a Boolean model into a program is thus to encode the active state of a variable with state 1 and the inactive state with 0. If the Boolean functions are represented as disjunctive normal forms, the clauses can be considered as a set of Boolean atoms of the form v or $\neg$v. Each clause c of the DNF of a variable v can directly be converted into a rule R such that, $\mathrm{head}(R) = \mathrm{v}_t^1$ and $\forall v'^{t-1} \in \mathcal{F}$, $\mathrm{v}'^{1}_{t-1} \in \mathrm{body}(R) \iff \mathrm{v}' \in c$ and $\mathrm{v}'^{0}_{t-1} \in \mathrm{body}(R) \iff (\neg \mathrm{v}') \in c$. Finally, the following default function allows to force the variables back to 0 when the original Boolean function should not be true:

$$d : \mathcal{S}^{\mathcal{F}} \times \wp(\mathcal{T}) \to \wp(\mathcal{A}|_{\mathcal{T}})$$
$$(s, Z) \mapsto \{v_t^0 \mid v_t \in Z\}$$

In Definition 15, we formalize the asynchronous semantics that imposes that no more than one regular variable can change its value in each transition. The observation variables are not counted since they have no equivalent in feature variables to be compared to. As for the previous synchronous semantics, we use here a "pool" of atoms, made of rules conclusions and default atoms, that may be used to build the next states. The default function d used here is inspired from the previous synchronous semantics, with an additional constraint: its result always contains the atoms of the initial state. Constrains are also added on the next state to limit to at most one regular variable change. Moreover, contrary to the synchronous semantics, the asynchronous semantics prioritizes the changes. Thus, for a self-transition $(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ with

$\mathsf{sp}_{\overline{\mathcal{T}}\to\overline{\mathcal{F}}}(s') \subseteq s$ to occur, it is required that all atoms of regular variables in the pool are in $\mathsf{sp}_{\overline{\mathcal{F}}\to\overline{\mathcal{T}}}(s)$, i.e., this only happens when (s, s') is a point attractor, in the sense that all regular variables cannot change their value.

Definition 15 (Asynchronous semantics) Let $d \in (\mathcal{S}^{\mathcal{F}} \times \wp(\mathcal{T}) \to \wp(\mathcal{A}|_{\mathcal{T}}))$, so that $\forall s \in \mathcal{S}^{\mathcal{F}}, \forall W \subseteq \mathcal{T}, W \subseteq \mathrm{var}(d(s, W)) \wedge \mathsf{sp}_{\overline{\mathcal{F}}\to\overline{\mathcal{T}}}(s) \subseteq d(s, \emptyset) \subseteq d(s, W)$. The *asynchronous semantics* $\mathcal{T}_{asyn}$ is defined by:

$$\begin{aligned}\mathcal{T}_{asyn} : P \mapsto \{(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}} \mid\ & s' \subseteq \mathsf{Conclusions}(s, P) \cup \\ & d(s, \mathcal{T} \setminus \mathrm{var}(\mathsf{Conclusions}(s, P))) \wedge \\ & \big(|\mathsf{sp}_{\overline{\mathcal{F}}\to\overline{\mathcal{T}}}(s) \setminus s'| = 1 \vee \mathsf{Conclusions}|_{\overline{\mathcal{T}}}(s, P) \cup \\ & \quad d_{\overline{\mathcal{T}}}(s, \overline{\mathcal{T}} \setminus \mathrm{var}(\mathsf{Conclusions}(s, P))) = \mathsf{sp}_{\overline{\mathcal{F}}\to\overline{\mathcal{T}}}(s)\big)\}\end{aligned}$$

where the notations $\mathcal{A}|_{\mathcal{T}}$, $\mathsf{Conclusions}|_{\overline{\mathcal{T}}}$ and $d|_{\overline{\mathcal{T}}}$ come from Definition 12.

A typical mapping for d is: $d : (s, W) \mapsto \mathsf{sp}_{\overline{\mathcal{F}}\to\overline{\mathcal{T}}}(s) \cup O$, where O is a set of atoms on observation variables with arbitrary values, thus conserving the previous values for regular variables and ignoring the second argument.

In Definition 16, we formalize the general semantics as a more permissive version of the synchronous one: any subset of the variables can change their value in a transition. This semantics uses the same "pool" of atoms than the synchronous semantics containing rules conclusions of P and default atoms provided by d, and no constraint. However, as for the asynchronous semantics, the atoms of the initial state must always be featured as default atoms. Thus, a self-transition $(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ with $\mathsf{sp}_{\overline{\mathcal{F}}\to\overline{\mathcal{T}}}(s) \subseteq s'$ occurs for each state s because, intuitively, the empty set of variables can always be selected for update. However, as for the synchronous semantics, such a self-transition is a point attractor only if all atoms of regular variables in the "pool" are in $\mathsf{sp}_{\overline{\mathcal{F}}\to\overline{\mathcal{T}}}(s)$. Finally, we note that the general semantics contains the dynamics of both the synchronous and the asynchronous semantics, but also other dynamics not featured in these two other semantics.

Definition 16 (General semantics) Let $d \in (\mathcal{S}^{\mathcal{F}} \times \wp(\mathcal{T}) \to \wp(\mathcal{A}|_{\mathcal{T}}))$, so that $\forall s \in \mathcal{S}^{\mathcal{F}}, \forall W \subseteq \mathcal{T}, W \subseteq \mathrm{var}(d(s, W)) \wedge \mathsf{sp}_{\overline{\mathcal{F}}\to\overline{\mathcal{T}}}(s) \subseteq d(s, \emptyset) \subseteq d(s, W)$. The *general semantics* $\mathcal{T}_{gen}$ is defined by:

$$\begin{aligned}\mathcal{T}_{gen} : P \mapsto \{(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}} \mid\ & s' \subseteq \mathsf{Conclusions}(s, P) \cup \\ & d(s, T \setminus \mathrm{var}(\mathsf{Conclusions}(s, P)))\}.\end{aligned}$$

Figure 5 gives an example of the transitions corresponding to these three semantics on a simple Boolean network of two variables inhibiting each other. The corresponding optimal $\mathcal{DM}$VLP is given below each transition graph. In this example, the three programs share the rules corresponding to the inhibitions: $a_t^0 \leftarrow b_{t-1}^1$ and $a_t^1 \leftarrow b_{t-1}^0$ model the inhibition of a by b, while $b_t^0 \leftarrow a_{t-1}^1$ and $b_t^1 \leftarrow a_{t-1}^0$ model the inhibition of b by a. However, generally speaking, there may not always exist such shared rules, for instance if the interactions they represent are somehow ignored by the semantics behavior.

Furthermore, in this example, we observe additional rules (w.r.t. the synchronous case) that appear in both the asynchronous and general semantics cases. Those rules capture the default behavior of both semantics, that is, the projection of the feature state as possible target atoms. Again, such rules may not appear generally speaking, because the dynamics of the system might combine with the dynamics semantics, thus possibly merging multiple rules into more general ones (for example, conservation rules becoming rules with an empty body).

Example 15 As for the synchronous semantics, it is possible to reproduce classical Boolean network dynamics using the asynchronous ($\mathcal{T}_{asyn}$) and general semantics ($\mathcal{T}_{gen}$) with the same encoding of rules, and a similar default function where the projection of the current state is added:

$$\begin{aligned} d : \mathcal{S}^{\mathcal{F}} \times \wp(\mathcal{T}) &\to \wp(\mathcal{A}|_{\mathcal{T}}) \\ (s, Z) &\mapsto \{v_t^0 \mid v_t \in Z\} \cup \mathsf{sp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(s) \end{aligned}$$

Finally, with Theorem 2, we state that the definitions and method developed in the previous section are independent of the chosen semantics as long as it respect Theorem 1.

Theorem 2 (Semantics-Free Correctness) *Let P be a $\mathcal{DM}$*VLP*.*

- $\mathcal{T}_{syn}(P) = \mathcal{T}_{syn}(P_{\mathcal{O}}(\mathcal{T}_{syn}(P)))$,
- $\mathcal{T}_{asyn}(P) = \mathcal{T}_{asyn}(P_{\mathcal{O}}(\mathcal{T}_{asyn}(P)))$,
- $\mathcal{T}_{gen}(P) = \mathcal{T}_{gen}(P_{\mathcal{O}}(\mathcal{T}_{gen}(P)))$.

The next section focuses on methods and algorithm to learn the optimal program.

4 GULA

In Algorithm 1 we presented a trivial algorithm to obtain the optimal program. In this section we present a more efficient algorithm based on inductive logic programming.

Until now, the LF1T algorithm [20,45,47] only tackled the learning of synchronous deterministic programs. Using the formalism introduced in the previous sections, it can now be revised to learn systems from transitions produced from any semantics respecting Theorem 1 like the three semantics defined above. Furthermore, both deterministic and non-deterministic systems can now be learned.

4.1 Learning operations

This section focuses on the manipulation of programs for the learning process. Definition 17 and Definition 18 formalize the main atomic operations

performed on a rule or a program by the learning algorithm, whose objective is to make minimal modifications to a given $\mathcal{DM}$VLP in order to be consistent with a new set of transitions.

Definition 17 (Rule least specialization) Let R be a $\mathcal{M}$VL rule and $s \in \mathcal{S}^{\mathcal{F}}$ such that $R \sqcap s$. The *least specialization* of R by s according to $\mathcal{F}$ and $\mathcal{A}$ is:

$$L_{\text{spe}}(R, s, \mathcal{A}, \mathcal{F}) := \{\text{head}(R) \leftarrow \text{body}(R) \cup \{\text{v}^{val}\} \mid \text{v} \in \mathcal{F} \wedge \text{v}^{val} \in \mathcal{A} \wedge \text{v}^{val} \notin s \wedge \forall val' \in \mathbb{N}, \text{v}^{val'} \notin \text{body}(R)\}.$$

The least specialization $L_{\text{spe}}(R, s, \mathcal{A}, \mathcal{F})$ produces a set of rule which matches all states that R matches except s. Thus $L_{\text{spe}}(R, s, \mathcal{A}, \mathcal{F})$ realizes all transitions that R realizes except the ones starting from s. Note that $\forall R \in P, R \sqcap s \wedge |\text{body}(R)| = |\mathcal{F}| \implies L_{\text{spe}}(R, s, \mathcal{A}, \mathcal{F}) = \emptyset$, i.e., a rule R matching s cannot be modified to make it not match s if its body already contains all feature variables, because nothing can be added in its body.

Example 16 Let $\mathcal{F} := \{a_{t-1}, b_{t-1}, c_{t-1}\}$ and $\mathsf{dom}(a_{t-1}) := \{0, 1\}, \mathsf{dom}(b_{t-1}) := \{0, 1, 2\}, \mathsf{dom}(c_{t-1}) := \{0, 1, 2, 3\}$. We give below three examples of least specialization on different initial rules and states. These situations could very well happen in the learning of a same set of transitions, at different steps of the process. The added conditions are highlighted in bold.

$L_{\text{spe}}(a_t^0 \leftarrow \emptyset,$
$\{a_{t-1}^0, b_{t-1}^1, c_{t-1}^2\}, \mathcal{A}, \mathcal{F}) = \{$
$a_t^0 \leftarrow \mathbf{a_{t-1}^1},$
$a_t^0 \leftarrow \mathbf{b_{t-1}^0},$
$a_t^0 \leftarrow \mathbf{b_{t-1}^2},$
$a_t^0 \leftarrow \mathbf{c_{t-1}^0},$
$a_t^0 \leftarrow \mathbf{c_{t-1}^1},$
$a_t^0 \leftarrow \mathbf{c_{t-1}^3}\ \}$

$L_{\text{spe}}(b_t^0 \leftarrow b_{t-1}^1,$
$\{a_{t-1}^0, b_{t-1}^1, c_{t-1}^2\}, \mathcal{A}, \mathcal{F}) = \{$
$b_t^0 \leftarrow \mathbf{a_{t-1}^1} \wedge b_{t-1}^1,$
$b_t^0 \leftarrow b_{t-1}^1 \wedge \mathbf{c_{t-1}^0},$
$b_t^0 \leftarrow b_{t-1}^1 \wedge \mathbf{c_{t-1}^1},$
$b_t^0 \leftarrow b_{t-1}^1 \wedge \mathbf{c_{t-1}^3}\ \}$

$L_{\text{spe}}(c_t^0 \leftarrow a_{t-1}^0 \wedge c_{t-1}^2,$
$\{a_{t-1}^0, b_{t-1}^1, c_{t-1}^2\}, \mathcal{A}, \mathcal{F}) = \{$
$c_t^0 \leftarrow a_{t-1}^0 \wedge \mathbf{b_{t-1}^0} \wedge c_{t-1}^2,$
$c_t^0 \leftarrow a_{t-1}^0 \wedge \mathbf{b_{t-1}^2} \wedge c_{t-1}^2\ \}$

For $a_t^0 \leftarrow \emptyset$, the rule having an empty body, all possible variable values (given by dom) not appearing in the given state are candidate for a new condition. For $b_t^2 \leftarrow b_{t-1}^1$, there is a condition on b in the body, therefore only conditions on a and c can be added. For $c_t^3 \leftarrow a_{t-1}^1 \wedge c_{t-1}^3$, only conditions on b can be added. Finally we can consider a case like $L_{\text{spe}}(a_t^1 \leftarrow a_{t-1}^0 \wedge b_{t-1}^1 \wedge c_{t-1}^2, \{a_{t-1}^0, b_{t-1}^1, c_{t-1}^2\}, \mathcal{A}, \mathcal{F}) = \emptyset$ where a condition already exists for each variable and thus no minimal specialization of the body can be produced, thus resulting in an empty set of rules.

Definition 18 (Program least revision) Let P be a $\mathcal{DM}$VLP, $s \in \mathcal{S}^{\mathcal{F}}$ and $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ such that $\text{first}(T) = \{s\}$. Let $R_P := \{R \in P \mid R \text{ conflicts with } T\}$. The *least revision* of P by T according to $\mathcal{A}$ and $\mathcal{F}$ is $L_{\text{rev}}(P, T, \mathcal{A}, \mathcal{F}) := (P \setminus R_P) \cup \bigcup_{R \in R_P} L_{\text{spe}}(R, s, \mathcal{A}, \mathcal{F})$.

Note that according to Definition 18, $\text{first}(T) = \{s\}$ implies that all transitions for T have s as initial state.

Example 17 Let $\mathcal{F} := \{a_{t-1}, b_{t-1}, c_{t-1}\}$ and $\mathsf{dom}(a_{t-1}) := \{0,1\}, \mathsf{dom}(b_{t-1}) := \{0,1,2\}, \mathsf{dom}(c_{t-1}) := \{0,1,2,3\}$. Let T be as set of transitions and P a $\mathcal{DM}$VLP as follows.

$T := \{$
$(\{a^0_{t-1}, b^1_{t-1}, c^2_{t-1}\}, \{a^1_t, b^1_t, c^2_t\}),$
$(\{a^0_{t-1}, b^1_{t-1}, c^2_{t-1}\}, \{a^0_t, b^2_t, c^2_t\}),$
$(\{a^0_{t-1}, b^1_{t-1}, c^2_{t-1}\}, \{a^0_t, b^1_t, c^1_t\}),$
$(\{a^0_{t-1}, b^1_{t-1}, c^2_{t-1}\}, \{a^0_t, b^1_t, c^3_t\}),$
$\}$

$P := \{$
$a^0_t \leftarrow \emptyset,$
$a^1_t \leftarrow \emptyset,$
$\mathbf{b^0_t \leftarrow b^1_{t-1}},$
$b^1_t \leftarrow \emptyset,$
$\mathbf{c^0_t \leftarrow a^0_{t-1} \wedge b^1_{t-1} \wedge c^2_{t-1}},$
$\mathbf{c^0_t \leftarrow c^2_{t-1}},$
$c^1_t \leftarrow a^0_{t-1},$
$c^2_t \leftarrow a^1_{t-1},$
$c^2_t \leftarrow b^1_{t-1},$
$c^3_t \leftarrow c^2_{t-1}$ $\}$

$L_{\text{rev}}(P, T, \mathcal{A}, \mathcal{F}) := \{$
$a^0_t \leftarrow \emptyset,$
$a^1_t \leftarrow \emptyset,$
$b^0_t \leftarrow \mathbf{a^1_{t-1}} \wedge b^1_{t-1},$
$b^0_t \leftarrow b^1_{t-1} \wedge \mathbf{c^0_{t-1}},$
$b^0_t \leftarrow b^1_{t-1} \wedge \mathbf{c^1_{t-1}},$
$b^0_t \leftarrow b^1_{t-1} \wedge \mathbf{c^2_{t-1}},$
$b^1_t \leftarrow \emptyset,$
$c^0_t \leftarrow \mathbf{a^1_{t-1}} \wedge c^2_{t-1},$
$c^0_t \leftarrow \mathbf{b^0_{t-1}} \wedge c^2_{t-1},$
$c^0_t \leftarrow \mathbf{b^2_{t-1}} \wedge c^2_{t-1},$
$c^1_t \leftarrow a^0_{t-1},$
$c^2_t \leftarrow a^1_{t-1},$
$c^2_t \leftarrow b^1_{t-1},$
$c^3_t \leftarrow c^2_{t-1}$ $\}$

Here, we have $\text{first}(T) = \{\{a^0_{t-1}, b^1_{t-1}, c^2_{t-1}\}\}$ and thus the least revision of Definition 18 can be applied on P. Moreover, $R_P = \{b^0_t \leftarrow b^1_{t-1}, c^0_t \leftarrow a^0_{t-1} \wedge b^1_{t-1} \wedge c^2_{t-1}, c^0_t \leftarrow c^2_{t-1}\}$; these rules are highlighted in bold in P. The least revision of P by T over $\mathcal{A}$ and $\mathcal{F}$, $L_{\text{rev}}(P, T, \mathcal{A}, \mathcal{F})$, is obtained by removing the rules of R_P from P and adding their least specialization, added conditions are in bold in $L_{\text{rev}}(P, T, \mathcal{A}, \mathcal{F})$ and are detailed in Example 16, except for $a^0_t \leftarrow \emptyset$ which does not need to be revised because it is consistent with T since a^0_t is observed in some target states.

Theorem 3 states properties on the least revision, in order to prove it suitable to be used in the learning algorithm.

Theorem 3 (Properties of Least Revision) *Let R be a $\mathcal{M}$VL rule and $s \in \mathcal{S}^{\mathcal{F}}$ such that $R \sqcap s$. Let $S_R := \{s' \in \mathcal{S}^{\mathcal{F}} \mid R \sqcap s'\}$ and $S_{\text{spe}} := \{s' \in \mathcal{S}^{\mathcal{F}} \mid \exists R' \in L_{\text{spe}}(R, s, \mathcal{A}, \mathcal{F}), R' \sqcap s'\}$.*

Let P be a $\mathcal{DM}$VLP and $T, T' \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ such that $|\text{first}(T)| = 1 \wedge \text{first}(T) \cap \text{first}(T') = \emptyset$. The following results hold:

1. $S_{\text{spe}} = S_R \setminus \{s\}$,
2. $L_{\text{rev}}(P, T, \mathcal{A}, \mathcal{F})$ *is consistent with* T,
3. $\overset{P}{\hookrightarrow} T' \implies \overset{L_{\text{rev}}(P,T,\mathcal{A},\mathcal{F})}{\hookrightarrow} T'$,
4. $\overset{P}{\hookrightarrow} T \implies \overset{L_{\text{rev}}(P,T,\mathcal{A},\mathcal{F})}{\hookrightarrow} T$,
5. P *is complete* $\implies L_{\text{rev}}(P, T, \mathcal{A}, \mathcal{F})$ *is complete.*

The next properties are directly used in the learning algorithm. Proposition 2 gives an explicit definition of the optimal program for an empty set of transitions, which is the starting point of the algorithm. Proposition 3 gives a method to obtain the optimal program from any suitable program by simply removing the dominated rules; this means that the $\mathcal{DM}$VLP optimal for a set of transitions can be obtained from any $\mathcal{DM}$VLP suitable for the same set of transitions by removing all the dominated rules. Finally, in association with these two results, Theorem 4 gives a method to iteratively compute $P_{\mathcal{O}}(T)$ for any $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, starting from $P_{\mathcal{O}}(\emptyset)$.

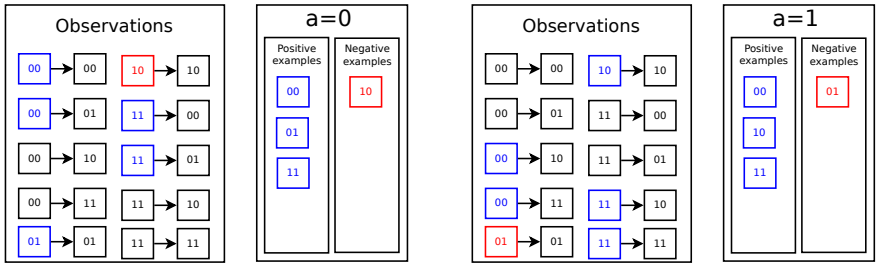


Fig. 6: Preprocessing of the general semantics state transitions of Figure 5 (right) into positive/negative example of the occurence of each value of variable a in next state. In blue (resp. red) are positives (resp. negatives) examples of the occurence of a_t^0 (left) and a_t^1 (right) in next state.

Proposition 2 (Optimal Program of Empty Set) $P_{\mathcal{O}}(\emptyset) = \{\mathrm{v}^{val} \leftarrow \emptyset \mid \mathrm{v} \in \mathcal{T} \wedge \mathrm{v}^{val} \in \mathcal{A}|_{\mathcal{T}}\}$.

Proposition 3 (From Suitable to Optimal) *Let* $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. *If* P *is a* $\mathcal{DM}$VLP *suitable for* T, *then* $P_{\mathcal{O}}(T) = \{R \in P \mid \forall R' \in P, R' \geq R \implies R' = R\}$.

Theorem 4 (Least Revision and Suitability) *Let* $s \in \mathcal{S}^{\mathcal{F}}$ *and* $T, T' \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ *such that* $|\mathrm{first}(T')| = 1 \wedge \mathrm{first}(T) \cap \mathrm{first}(T') = \emptyset$. $L_{\mathrm{rev}}(P_{\mathcal{O}}(T), T', \mathcal{A}, \mathcal{F})$ *is a* $\mathcal{DM}$VLP *suitable for* $T \cup T'$.

4.2 Algorithm

In this section we present **GULA**: the General Usage LFIT Algorithm, a revision of the **LF1T** algorithm [20,45] to capture a set of multi-valued dynamics that especially encompass the classical synchronous, asynchronous and general semantics dynamics. For this learning algorithm to operate, there is no restriction on the semantics. **GULA** learns the optimal program that, under the same semantics, is able to exactly reproduce a complete set of observations, if the semantics respect Theorem 1. Section 5 will be devoted to also learning the behaviors of the semantics itself, if it is unknown.

GULA learns a logic program from the observations of its state transitions. Given as input a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, **GULA** iteratively constructs a $\mathcal{DM}$VLP that models the dynamics of the observed system by applying the method formalized in the previous section as shown in Algorithm 2. The algorithm can be used for both learning possibility or impossibility depending of its parameter *learning_mode*. When learning possibility (*learning_mode* = "*possibility*"), the algorithm will learn the optimal logic program $P_{\mathcal{O}}(T)$ and this is what will be discussed in this section. The second mode is used in a heuristical approach to obtain predictive model from partial observation and will be discussed in later sections.

Algorithm 2 GULA

- **INPUT:** a set of atoms $\mathcal{A}'$, a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}'} \times \mathcal{S}^{\mathcal{T}'}$, two sets of variables $\mathcal{F}'$ and $\mathcal{T}'$, a string *learning_mode* $\in$ {"*possibility*", "*impossibility*"}.
- For each atom $\mathrm{v}^{val} \in \mathcal{A}'$ of each variable $\mathrm{v} \in \mathcal{T}'$:
  - if *learning_mode* = "*possibility*":
    - Extract all states from which transition to v^{val} **does not** exist: $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$.
  - if *learning_mode* = "*impossibility*":
    - Extract all states from which transition to v^{val} **do** exist: $Neg_{\mathrm{v}^{val}} := \{s \mid \exists (s, s') \in T, \mathrm{v}^{val} \in s'\}$.
  - Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \emptyset\}$.
  - For each state $s \in Neg_{\mathrm{v}^{val}}$:
    - Extract and remove the rules of $P_{\mathrm{v}^{val}}$ that match s: $M_{\mathrm{v}^{val}} := \{R \in P \mid \mathrm{body}(R) \subseteq s\}$ and $P_{\mathrm{v}^{val}} := P_{\mathrm{v}^{val}} \setminus M_{\mathrm{v}^{val}}$.
    - $LS := \emptyset$
    - For each rule $R \in M_{\mathrm{v}^{val}}$:
      - Compute its least specialization $P' = L_{\mathrm{spe}}(R, s, \mathcal{A}', \mathcal{F}')$.
      - Remove all the rules in P' dominated by a rule in $P_{\mathrm{v}^{val}}$.
      - Remove all the rules in P' dominated by a rule in LS.
      - Remove all the rules in LS dominated by a rule in P'.
      - $LS := LS \cup P'$.
    - Add all remaining rules of LS to $P_{\mathrm{v}^{val}}$: $P_{\mathrm{v}^{val}} := P_{\mathrm{v}^{val}} \cup LS$.
  - $P := P \cup P_{\mathrm{v}^{val}}$
- **OUTPUT:** P (P is $P_{\mathcal{O}}(T)$ if *learning_mode* = "*possibility*").

From the set of transitions T, **GULA** learns the conditions under which each $\mathrm{v}^{val} \in \mathcal{A}' \subseteq \mathcal{A}, \mathrm{v} \in \mathcal{T}' \subseteq \mathcal{T}$ may appear in the next state. The algorithm starts by computing the set of all negative examples of the appearance of v^{val} in next state: all states such that v never takes the value *val* in the next state of a transition of T. Those negative examples are then used during the following learning phase to iteratively learn the set of rules $P_{\mathcal{O}}(T)$. The learning phase starts by initializing a set of rules $P_{\mathrm{v}^{val}}$ to $\{R \in P_{\mathcal{O}}(\emptyset) \mid \mathrm{head}(R) = \mathrm{v}^{val}\} = \{\mathrm{v}^{val} \leftarrow \emptyset\}$. $P_{\mathrm{v}^{val}}$ is iteratively revised against each negative example *neg* in $Neg_{\mathrm{v}^{val}}$. All rules R_m of $P_{\mathrm{v}^{val}}$ that match *neg* have to be revised. In order for $P_{\mathrm{v}^{val}}$ to remain optimal, the revision of each R_m must not match *neg* but still matches every other state that R_m matches. To ensure that, the least specialization (see Definition 17) is used to revise each conflicting rule R_m. For each variable of $\mathcal{F}'$ so that $\mathrm{body}(R_m)$ has no condition over it, a condition over another value than the one observed in state *neg* can be added. None of those revision match *neg* and all states matched by R_m are still matched by at least one of its revisions. Each revised rule can be dominated by a rule in $P_{\mathrm{v}^{val}}$ or another revised rules and thus dominance must be checked from both. The non-dominated revised rules are then added to $P_{\mathrm{v}^{val}}$. Once $P_{\mathrm{v}^{val}}$ has been revised against all negatives example of $Neg_{\mathrm{v}^{val}}$, $P_{\mathrm{v}^{val}} = \{R \in P_{\mathcal{O}}(T) \mid \mathrm{head}(R) = \mathrm{v}^{val}\}$. Finally, $P_{\mathrm{v}^{val}}$ is added to P and the loop restarts

with another atom. Once all values of each variable have been treated, the algorithm outputs P which is then equal to $P_{\mathcal{O}}(T)$. More discussion of the implementation and detailed pseudocode are given in appendix. The source code of the algorithm is available at `https://github.com/Tony-sama/pylfit` under GPL-3.0 License.

Example 18 Execution of $\mathbf{GULA}(\mathcal{A}, T, \mathcal{F}, \mathcal{T})$ on the synchronous state transitions of Figure 5 (left):

- $\mathcal{F} = \{a_{t-1}, b_{t-1}\}$,
- $\mathcal{T} = \{a_t, b_t\}$,
- $\mathcal{A} = \{a^0_{t-1}, b^0_{t-1}, a^1_{t-1}, b^1_{t-1}, a^0_t, b^0_t, a^1_t, b^1_t\}$
- $T = \{ (\{a^0_{t-1}, b^0_{t-1}\}, \{a^1_t, b^1_t\}), (\{a^0_{t-1}, b^1_{t-1}\}, \{a^0_t, b^1_t\}), (\{a^1_{t-1}, b^0_{t-1}\}, \{a^1_t, b^0_t\}), (\{a^1_{t-1}, b^1_{t-1}\}, \{a^0_t, b^0_t\}) \}$

Table 1 provides each $Neg_{\mathrm{v}^{val}}$ (first column) and shows the iterative evolution of $P_{\mathrm{v}^{val}}$ (last column) over each $neg \in Neg_{\mathrm{v}^{val}}$ during the execution of $\mathbf{GULA}(\mathcal{A}, T, \mathcal{F}, \mathcal{T})$. Rules in red in $P_{\mathrm{v}^{val}}$ of previous step match the current negative example neg and must be revised, while rules in blue in the last column dominate rules in blue produced by the least specialization (third column).

Table 1: Iterative evolution of $P_{\mathrm{v}^{val}}$ over each element of $Neg_{\mathrm{v}^{val}}$ for each $\mathrm{v}^{val} \in \mathcal{A}|_{\mathcal{T}}$ during the execution of $\mathbf{GULA}(\mathcal{A}, T, \mathcal{F}, \mathcal{T})$ over the transitions of Figure 5 (left).

- $Neg_{a^0_t} = \{\{a^0_{t-1}, b^0_{t-1}\}, \{a^1_{t-1}, b^0_{t-1}\}\}, P_{a^0_t} = \{a^0_t \leftarrow \emptyset\}$

$neg \in Neg_{a^0_t}$	M	Least specializations	$P_{a^0_t}$
(a^0_{t-1}, b^0_{t-1})	$\{a^0_t \leftarrow \emptyset\}$	$\{a^0_t \leftarrow a^1_{t-1}, a^0_t \leftarrow b^1_{t-1}\}$	$\{a^0_t \leftarrow a^1_{t-1}, a^0_t \leftarrow b^1_{t-1}\}$
(a^1_{t-1}, b^0_{t-1})	$\{a^0_t \leftarrow a^1_{t-1}\}$	$\{a^0_t \leftarrow a^1_{t-1} \wedge b^1_{t-1}.\}$	$\{a^0_t \leftarrow b^1_{t-1}\}$

- $Neg_{a^1_t} = \{\{a^0_{t-1}, b^1_{t-1}\}, \{a^1_{t-1}, b^1_{t-1}\}\}, P_{a^1_t} = \{a^1_t \leftarrow \emptyset\}$

$neg \in Neg_{a^1_t}$	M	Least specializations	$P_{a^1_t}$
(a^0_{t-1}, b^1_{t-1})	$\{a^1_t \leftarrow \emptyset\}$	$\{a^1_t \leftarrow a^1_{t-1}, a^1_t \leftarrow b^0_{t-1}\}$	$\{a^1_t \leftarrow a^1_{t-1}, a^1_t \leftarrow b^0_{t-1}\}$
(a^1_{t-1}, b^1_{t-1})	$\{a^1_t \leftarrow a^1_{t-1}\}$	$\{a^1_t \leftarrow a^1_{t-1} \wedge b^0_{t-1}\}$	$\{a^1_t \leftarrow b^0_{t-1}\}$

- $Neg_{b^0_t} = \{\{a^0_{t-1}, b^1_{t-1}\}, \{a^0_{t-1}, b^0_{t-1}\}\}, P_{b^0_t} = \{b^0_t \leftarrow \emptyset\}$

$neg \in Neg_{b^0_t}$	M	Least specializations	$P_{b^0_t}$
(a^0_{t-1}, b^1_{t-1})	$\{b^0_t \leftarrow \emptyset\}$	$\{b^0_t \leftarrow a^1_{t-1}, b^0_t \leftarrow b^0_{t-1}\}$	$\{b^0_t \leftarrow a^1_{t-1}, b^0_t \leftarrow b^0_{t-1}\}$
(a^0_{t-1}, b^0_{t-1})	$\{b^0_t \leftarrow b^0_{t-1}\}$	$\{b^0_t \leftarrow a^1_{t-1} \wedge b^0_{t-1}\}$	$\{b^0_t \leftarrow a^1_{t-1}\}$

- $Neg_{b^1_t} = \{\{a^1_{t-1}, b^0_{t-1}\}, \{a^1_{t-1}, b^1_{t-1}\}\}, P_{b^1_t} = \{b^1_t \leftarrow \emptyset\}$

$neg \in Neg_{b^1_t}$	M	Least specializations	$P_{b^1_t}$
(a^1_{t-1}, b^0_{t-1})	$\{b^1_t \leftarrow \emptyset\}$	$\{b^1_t \leftarrow a^0_{t-1}, b^1_t \leftarrow b^1_{t-1}\}$	$\{b^1_t \leftarrow a^0_{t-1}, b^1_t \leftarrow b^1_{t-1}\}$
(a^1_{t-1}, b^1_{t-1})	$\{b^1_t \leftarrow b^1_{t-1}\}$	$\{b^1_t \leftarrow a^0_{t-1} \wedge b^1_{t-1}\}$	$\{b^0_t \leftarrow a^1_{t-1}\}$

Example 19 Execution of $\mathbf{GULA}(\mathcal{A}, T, \mathcal{F}, \mathcal{T})$ on the asynchronous state transitions of Figure 5 (middle):

- $\mathcal{F} = \{a_{t-1}, b_{t-1}\}$,
- $\mathcal{T} = \{a_t, b_t\}$,

- $\mathcal{A} = \{a^0_{t-1}, b^0_{t-1}, a^1_{t-1}, b^1_{t-1}, a^0_t, b^0_t, a^1_t, b^1_t\}$
- $T = \{\ (\{a^0_{t-1}, b^0_{t-1}\}, \{a^0_t, b^1_t\}), (\{a^0_{t-1}, b^0_{t-1}\}, \{a^1_t, b^0_t\}), (\{a^0_{t-1}, b^1_{t-1}\}, \{a^0_t, b^1_t\}),$
  $(\{a^1_{t-1}, b^0_{t-1}\}, \{a^1_t, b^0_t\}), (\{a^1_{t-1}, b^1_{t-1}\}, \{a^0_t, b^0_t\})\ (\{a^1_{t-1}, b^1_{t-1}\}, \{a^1_t, b^1_t\})\ \}$

Table 2 provides each $Neg_{\mathrm{v}^{val}}$ (first column) and shows the iterative evolution of $P_{\mathrm{v}^{val}}$ (last column) over each $neg \in Neg_{\mathrm{v}^{val}}$ during the execution of **GULA**$(\mathcal{A}, T, \mathcal{F}, \mathcal{T})$. Rules in red in the last column ($P_{\mathrm{v}^{val}}$) match the current negative example neg and must be revised, while rules in blue in the last column dominate rules in blue produced by the least specialization (third column, next line). For the general semantics transitions of Figure 5 (right), the additional transitions that are observed compared to the asynchronous case do not alter any $Neg_{\mathrm{v}^{val}}$, thus the learning process is the same as in Table 2 resulting in the same output program.

Table 2: Example of the execution of **GULA**$(\mathcal{A}, T, \mathcal{F}, \mathcal{T})$ over the transitions of Figure 5 (right) and, equivalently, the transitions of Figure 5 (right). For each $\mathrm{v}^{val} \in \mathcal{A}|_{\mathcal{T}}$ is given the iterative evolution over each element of $Neg_{\mathrm{v}^{val}}$ (1^{st} col.) of the set of matching rules $M \subseteq P_{\mathrm{v}^{val}}$ (2^{nd} col.), their least specializations (3^{rd} col.) and $P_{\mathrm{v}^{val}}$ final state.

- $Neg_{a^0_t} = \{\{a^1_{t-1}, b^0_{t-1}\}\}, P_{a^0_t} = \{a^0_t \leftarrow \emptyset\}$

$neg \in Neg_{a^0_t}$	M	Least specializations	$P_{a^0_t}$
(a^1_{t-1}, b^0_{t-1})	$\{a^0_t \leftarrow \emptyset\}$	$\{a^0_t \leftarrow a^0_{t-1}, a^0_t \leftarrow b^1_{t-1}\}$	$\{a^0_t \leftarrow a^0_{t-1}, a^0_t \leftarrow b^1_{t-1}\}$

- $Neg_{a^1_t} = \{\{a^0_{t-1}, b^1_{t-1}\}\}, P_{a^1_t} = \{a^1_t \leftarrow \emptyset\}$

$neg \in Neg_{a^1_t}$	M	Least specializations	$P_{a^1_t}$
(a^0_{t-1}, b^1_{t-1})	$\{a^1_t \leftarrow \emptyset\}$	$\{a^1_t \leftarrow a^1_{t-1}, a^1_t \leftarrow b^0_{t-1}\}$	$\{a^1_t \leftarrow a^1_{t-1}, a^1_t \leftarrow b^0_{t-1}\}$

- $Neg_{b^0_t} = \{\{a^0_{t-1}, b^1_{t-1}\}\}, P_{b^0_t} = \{b^0_t \leftarrow \emptyset\}$

$neg \in Neg_{b^0_t}$	M	Least specializations	$P_{b^0_t}$
(a^0_{t-1}, b^1_{t-1})	$\{b^0_t \leftarrow \emptyset\}$	$\{b^0_t \leftarrow a^1_{t-1}, b^0_t \leftarrow b^0_{t-1}\}$	$\{b^0_t \leftarrow a^1_{t-1}, b^0_t \leftarrow b^0_{t-1}\}$

- $Neg_{b^1_t} = \{\{a^1_{t-1}, b^0_{t-1}\}\}, P_{b^1_t} = \{b^1_t \leftarrow \emptyset\}$

$neg \in Neg_{b^1_t}$	M	Least specializations	$P_{b^1_t}$
(a^1_{t-1}, b^0_{t-1})	$\{b^1_t \leftarrow \emptyset\}$	$\{b^1_t \leftarrow a^0_{t-1}, b^1_t \leftarrow b^1_{t-1}\}$	$\{b^1_t \leftarrow a^1_{t-1}, b^1_t \leftarrow b^1_{t-1}\}$

Theorem 5 gives the properties of the algorithm: **GULA** terminates and **GULA** is sounds, complete and optimal w.r.t. its input, i.e., all and only non-dominated consistent rules appear in its output program which is the optimal program of its input. Finally, Theorem 6 characterizes the algorithm time and memory complexities.

Theorem 5 (GULA Termination, Soundness, Completeness, Optimality) *Let* $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$.

(1) *Any call to* **GULA** *on finite sets terminates,*
(2) **GULA**$(\mathcal{A}, T, \mathcal{F}, \mathcal{T}) = P_{\mathcal{O}}(T)$,
(3) $\forall \mathcal{A}' \subseteq \mathcal{A}|_{\mathcal{T}}$, **GULA**$(\mathcal{A}_{\mathcal{F}} \cup \mathcal{A}', T, \mathcal{F}, \mathcal{T}) = \{R \in P_{\mathcal{O}}(T) \mid \mathrm{head}(R) \in \mathcal{A}'\}$.

Lemma 2 (Gula can learn from any pseudo-idempotent semantics) *Let DS be a pseudo-idempotent semantics, then from Theorem 1*

$$DS(\mathbf{GULA}(\mathcal{A}, DS(P), \mathcal{F}, \mathcal{T})) = DS(P_{\mathcal{O}}(DS(P))) = DS(P).$$

Lemma 2 is trivially provent from Theorem 5 since for any dynamical semantics DS and any $\mathcal{DM}$VLP P, $\mathbf{GULA}(\mathcal{A}, DS(P), \mathcal{F}, \mathcal{T}) = P_{\mathcal{O}}(DS(P))$.

Lemma 3 (Gula can learn from synchronous, asynchronous and general semantics)

- $\mathcal{T}_{syn}(\mathbf{GULA}(\mathcal{A}, \mathcal{T}_{syn}(P), \mathcal{F}, \mathcal{T})) = \mathcal{T}_{syn}(P_{\mathcal{O}}(\mathcal{T}_{syn}(P))) = \mathcal{T}_{syn}(P)$
- $\mathcal{T}_{asyn}(\mathbf{GULA}(\mathcal{A}, \mathcal{T}_{asyn}(P), \mathcal{F}, \mathcal{T})) = \mathcal{T}_{asyn}(P_{\mathcal{O}}(\mathcal{T}_{asyn}(P))) = \mathcal{T}_{asyn}(P)$
- $\mathcal{T}_{gen}(\mathbf{GULA}(\mathcal{A}, \mathcal{T}_{gen}(P), \mathcal{F}, \mathcal{T})) = \mathcal{T}_{gen}(P_{\mathcal{O}}(\mathcal{T}_{gen}(P))) = \mathcal{T}_{gen}(P)$

Lemma 3 is trivially proven from Theorem 2. Thus the algorithm can be used to learn from transitions produced from both synchronous, asynchronous and general semantics.

Theorem 6 (GULA Complexity) *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ be a set of transitions, Let $n := max(|\mathcal{F}|, |\mathcal{T}|)$ and $d := \max(\{|\mathsf{dom}(\mathrm{v})|) \in \mathbb{N} \mid \mathrm{v} \in \mathcal{F} \cup \mathcal{T}\}$. The worst-case time complexity of* **GULA** *when learning from T belongs to $\mathcal{O}(|T|^2 + |T| \times (2n^4 d^{2n+2} + 2n^3 d^{n+1}))$ and its worst-case memory use belongs to $\mathcal{O}(d^{2n} + 2nd^{n+1} + nd^{n+2})$.*

The worst case complexity of **GULA** is higher than the brute force algorithm of Algorithm 1. The brute force complexity is bound by the operation of removing the dominated rules ($O(nd^{2n+2})$), that also appear in **GULA**. This operation is done once in the brute force with all consistent rules and multiple time (for each negative example) in **GULA**, also **GULA** can generate several time the same rule. But, in practice, **GULA** is expected to manage much less rules than the whole set of possibility at each step since it remove dominated rules of previous step, thus globally dealing with less rules than all possibility and ending being more efficient in practice. Its scalability is evaluated in Section 7 with the brute force algorithm as baseline.

To use GULA for outputting predictions, we have to assume a semantics for the model. In the next section, we will exhibit an approach to avoid such a preliminary assumption and learn a whole system dynamics, including its semantics, in the form of a single propositional logic program.

5 Learning From Any Dynamical Semantics using Constraints

Any non-deterministic (and thus deterministic) discrete memory-less dynamical system can be represented by a $\mathcal{M}$VLP with some restrictions and a dedicated dynamical semantics. For this, programs must contain two types of rules: *possibility rules* which have conditions on variables at $t-1$ and conclusion on one variable at t, same as for $\mathcal{DM}$VLP; and *constraint rules* which have conditions on both t and $t-1$ but no conclusion. In the following, we also re-use the same notations as for the $\mathcal{M}$VL of Section 2.1 such as head(R), body(R) and var(head(R)).

5.1 Constraints $\mathcal{DM}$VLP

Definition 19 (Constrained $\mathcal{DM}$VLP) Let P' be a $\mathcal{DM}$VLP on $\mathcal{A}_{\mathsf{dom}}^{\mathcal{F}\cup\mathcal{T}}$, $\mathcal{F}$ and $\mathcal{T}$ two sets of variables, and ε a special variable with $\mathsf{dom}(\varepsilon) = \{0, 1\}$ so that $\varepsilon \notin \mathcal{T} \cup \mathcal{F}$. A $\mathcal{CDM}$VLP P is a $\mathcal{M}$VLP such that $P = P' \cup \{R \in \mathcal{M}\text{VL} \mid \text{head}(R) = \varepsilon^1 \wedge \forall \text{v}^{val} \in \text{body}(R), \text{v} \in \mathcal{F} \cup \mathcal{T}\}$. A $\mathcal{M}$VL rule R such that $\text{head}(R) = \varepsilon^1$ and $\forall \text{v}^{val} \in \text{body}(R), \text{v} \in \mathcal{F} \cup \mathcal{T}$ is called a $\mathcal{M}$VL *constraint*.

Moreover, in the following we denote $\mathcal{V} = \mathcal{F} \cup \mathcal{T} \cup \{\varepsilon\}$. This $\mathcal{V}$ is different than the one of P' (which is $\mathcal{F} \cup \mathcal{T}$, without the special variable ε). From now, a constraint C is denoted: $\overset{\perp}{\leftarrow} \text{body}(C)$.

Example 20 $\overset{\perp}{\leftarrow} a_t^0 \wedge a_{t-1}^0$ is a constraint that can prevent a to take the value 0 in two successive states. $\overset{\perp}{\leftarrow} b_t^1 \wedge d_t^2 \wedge c_{t-1}^2$ is a constraint that can prevent to have both b^1 and d^2 in the next state if c^2 appears in the initial state. $\overset{\perp}{\leftarrow} a_t^0 \wedge b_t^0$ is a constraint with only conditions in $\mathcal{T}$, it prevents a and b to take value 0 at same time. $\overset{\perp}{\leftarrow} a_{t-1}^0 \wedge b_{t-1}^0$ is a constraint with only conditions in $\mathcal{F}$, it prevents any transitions from a state where a and b have value 0, thus creating final states.

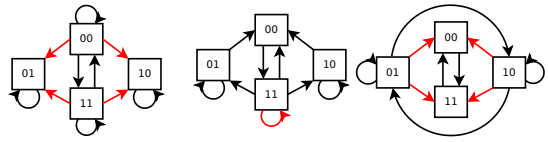
Definition 20 (Constraint-transition matching) Let $(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. The constraint C *matches* (s, s'), written $C \sqcap (s, s')$, iff $\text{body}(C) \subseteq s \cup s'$.

Using the notion of rule and constraint matching we can use a $\mathcal{CDM}$VLP to compute the next possible states. Definition 21 provides such a method based on synchronous semantic and constraints. Given a state, the set of possible next states is the Cartesian product of the conclusion of all matching rules and default atoms. Constraints rules are then used to discard states that would generate non-valid transitions.

Definition 21 (Synchronous constrained Semantics) The *synchronous constrained semantics* $\mathcal{T}_{syn-c}$ is defined by:

$$\mathcal{T}_{syn-c} : P \mapsto \{(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}} \mid s' \subseteq \mathsf{Conclusions}(s, P) \wedge \nexists C \in P, \text{head}(C) = \varepsilon^1 \wedge C \sqcap (s, s')\}$$

Figure 7 shows the dynamics of the Boolean network of Figure 5 under three semantics which dynamics cannot be reproduced using synchronous, asynchronous or general semantics on a program learned using **GULA**. In the first example (left), either all Boolean functions are applied simultaneously or nothing occurs (self-transition using projection). In the second example (center), the Boolean functions are applied synchronously but their is also always a possibility for any variable to take value 0 in the next state. In the third example (right), either the Boolean functions are applied synchronously, or each variable value is reversed (0 into 1 and 1 into 0). The original transitions of each dynamics are in black and the additional non-valid transitions in red.



// a := not b
$a_t^0 \leftarrow b_{t-1}^1$
$a_t^1 \leftarrow b_{t-1}^0$

// b := not a
$b_t^0 \leftarrow a_{t-1}^1$
$b_t^1 \leftarrow a_{t-1}^0$

// Stability rules
$a_t^0 \leftarrow a_{t-1}^0$
$a_t^1 \leftarrow a_{t-1}^1$
$b_t^0 \leftarrow b_{t-1}^0$
$b_t^1 \leftarrow b_{t-1}^1$

// Constraints
$\overset{\perp}{\leftarrow} a_t^0, b_t^1, b_{t-1}^0$
$\overset{\perp}{\leftarrow} a_t^1, b_t^0, a_{t-1}^0$
$\overset{\perp}{\leftarrow} a_t^1, b_t^0, b_{t-1}^1$
$\overset{\perp}{\leftarrow} a_t^0, b_t^1, a_{t-1}^1$

// a := not b
$a_t^0 \leftarrow b_{t-1}^1$
$a_t^1 \leftarrow b_{t-1}^0$

// b := not a
$b_t^0 \leftarrow a_{t-1}^1$
$b_t^1 \leftarrow a_{t-1}^0$

// Stability rules
$a_t^1 \leftarrow a_{t-1}^1$
$b_t^1 \leftarrow b_{t-1}^1$

// Degradation
$a_t^0 \leftarrow a_{t-1}^1$
$b_t^0 \leftarrow b_{t-1}^1$

// Constraints
$\overset{\perp}{\leftarrow} a_t^1, b_t^1, a_{t-1}^1$
$\overset{\perp}{\leftarrow} a_t^1, b_t^1, b_{t-1}^1$

// a := not b
$a_t^0 \leftarrow b_{t-1}^1$
$a_t^1 \leftarrow b_{t-1}^0$

// b := not a
$b_t^0 \leftarrow a_{t-1}^1$
$b_t^1 \leftarrow a_{t-1}^0$

// Inverse value
$a_t^0 \leftarrow a_{t-1}^1$
$a_t^1 \leftarrow a_{t-1}^0$
$b_t^0 \leftarrow b_{t-1}^1$
$b_t^1 \leftarrow b_{t-1}^0$

// Constraints
$\overset{\perp}{\leftarrow} a_t^1, b_t^1, a_{t-1}^1$
$\overset{\perp}{\leftarrow} a_t^0, b_t^0, a_{t-1}^0$
$\overset{\perp}{\leftarrow} a_t^1, b_t^1, b_{t-1}^1$
$\overset{\perp}{\leftarrow} a_t^0, b_t^0, b_{t-1}^0$

Fig. 7: States transitions diagrams corresponding to three semantics that do not respect Theorem 1 (in black) applied on the Boolean network of Figure 5. Using the synchronous semantics on the optimal program of the black transitions will produce in addition the red ones. Below each diagram, a $\mathcal{CDM}$VLP that can reproduce the same behavior using synchronous constrained semantics.

Using the original black transitions as input, **GULA** learns programs which, under the synchronous semantics (Definition 14), would realize the original black transitions plus the non-valid red ones. The idea is to learn constraints that would prevent those non-valid transitions to occur so that the observed dynamics is exactly reproduced using the synchronous constrained semantics of Definition 21. The $\mathcal{CDM}$VLP shown below each dynamics realize all original black transitions thanks to there rules and none of the red transitions thanks to their constraints.

Definition 22 (Conflict and Consistency of constraints) The constraint C *conflicts* with a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ when $\exists (s, s') \in T, C \sqcap (s, s')$. C is said to be *consistent* with T when C does not conflict with T.

Therefore, a constraint is consistent if it does not match any transitions of T.

Definition 23 (Complete set of constraints) A set of constraints SC is *complete* with a set of transitions T if $\forall (s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}, (s, s') \notin T \implies \exists C \in SC, C \sqcap (s, s')$.

Definition 24 groups all the properties that we want the learned set of constraints to have: suitability and optimality, and Proposition 4 states that the optimal set of constraints of a set of transitions is unique.

Definition 24 (Suitable and optimal constraints) Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. A set of $\mathcal{M}$VL constraints SC is *suitable* for T when:

- SC is consistent with T,
- SC is complete with T,
- for all constraints C not conflicting with T, there exists $C' \in P$ such that $C' \geq C$.

If in addition, for all $C \in SC$, all the constraint rules C' belonging to a set of constraints suitable for T are such that $C' \geq C$ implies $C \geq C'$, then SC is called *optimal*.

Proposition 4 *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. The optimal set of constraints for T is unique and denoted $C_{\mathcal{O}}(T)$.*

The subset of constraints of $C_{\mathcal{O}}(T)$ that prevent transitions permitted by $P_{\mathcal{O}}(T)$ but not observed in T from happening, or, in other terms, constraints that match transitions in $\mathcal{T}_{syn-c}(P_{\mathcal{O}}(T))) \setminus T$, is denoted $C'_{\mathcal{O}}(T)$ and given in Definition 25. All constraints of $C_{\mathcal{O}}(T)$ that are not in this set can never match a transition produced by $P_{\mathcal{O}}(T)$ with $\mathcal{T}_{syn-c}$ and can thus be considered useless. Finally, Theorem 7 shows that any set of transitions T can be reproduced, using synchronous constrained semantics of Definition 21 on the $\mathcal{CDM}$VLP $P_{\mathcal{O}}(T) \cup C'_{\mathcal{O}}(T)$.

Definition 25 (Useful Constraints) Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$.

$C'_{\mathcal{O}}(T) := \{C \in C_{\mathcal{O}}(T) \mid \exists (s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}, C \sqcap (s, s') \wedge s \xrightarrow{P_{\mathcal{O}}(T)} s'\}$.

Theorem 7 (Optimal $\mathcal{DM}$VLP and Constraints Correctness Under Synchronous Constrained Semantics) *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, it holds that $T = \mathcal{T}_{syn-c}(P_{\mathcal{O}}(T) \cup C'_{\mathcal{O}}(T))$.*

5.2 Algorithm

In previous sections we presented a modified version of **GULA**: the General Usage LFIT Algorithm from [43], which takes as arguments a different set of variables for conditions and conclusions of rules. This modification allows to use this modified algorithm to learn constraints and thus $\mathcal{CDM}$VLP.

Algorithm 3 show the **Synchronizer** algorithm, which given a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ will output $P_{\mathcal{O}}(T) \cup C'_{\mathcal{O}}(T)$ using **GULA** and the properties introduced in the previous section. With the new version of **GULA** it is possible to encode meaning in the transitions we give as input to the algorithm. The constraints we want to learn are technically rules whose head is ϵ^1 with conditions on both $\mathcal{F}$ and $\mathcal{T}$. It is sufficient to make the union of the two states of a

transition and feed it to **GULA** to make it learn such rules. Constraints should match when an impossible transition is generated by the rules of the optimal program of T. **GULA** learns from negative examples and negative examples of impossible transitions are just the possible transitions, thus the transitions observed in T. Using the set of transitions $T' := \{(s \cup s', \{\varepsilon^0\}) \mid (s, s') \in T\}$ we can use **GULA** to learn such constraints with $GULA(\mathcal{A} \cup \{\varepsilon^1\}, T', \mathcal{F} \cup \mathcal{T}, \{\varepsilon\})$. Note that ε, from the algorithmic viewpoint, is just a dummy variable used to make every transition of T' a negative example of ε^1 which will be the only head of the rule we will learn here. The program produced will contain a set of rules that match none of the initial states of T' and thus none of the transitions of T but matches all other possible transitions according to **GULA** properties. Their head being ε^1, those rules are actually constraints over T. Since all and only such minimal rules are output by this second call to **GULA**, it correspond to $C_{\mathcal{O}}(T)$, which prevent every transitions that are not in T to be produced using the constraint synchronous semantics. Finally, the non-essential constraints can be discarded following Definition 25 and finally $P_{\mathcal{O}}(T) \cup C'_{\mathcal{O}}(T)$ is output. The source code of the algorithm is available at `https://github.com/Tony-sama/pylfit` under GPL-3.0 License.

Algorithm 3 **Synchronizer**

- **INPUT:** a set of atoms $\mathcal{A}$, a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, two sets of variables $\mathcal{F}$ and $\mathcal{T}$.
  // 1) Learn what is possible locally in a transition using GULA
- $P := GULA(\mathcal{A}, T, \mathcal{F}, \mathcal{T})$
  // 2) Encode negative examples of constraints, i.e., observed transitions
- Let ε be a special variable not in the system: $\varepsilon \notin \mathcal{F} \cup \mathcal{T}$
- $T' := \{(s \cup s', \{\varepsilon^0\}) \mid (s, s') \in T\}$
  // 3) Learn what is impossible state-wise in form of constraint using GULA
- $P' := GULA(\mathcal{A}|_{\mathcal{F} \cup \mathcal{T} \cup \{\varepsilon^1\}}, T', \mathcal{F} \cup \mathcal{T}, \{\varepsilon\})$
  // 4) Keep only applicable constraints
- $P'' := \emptyset$
- For each $C \in P'$
  // 4.1) Extract compatible rules
  - $C_{targets} := \{\mathrm{v} \in \mathcal{T} \mid \exists val \in \mathsf{dom}(\mathrm{v}), \mathrm{v}^{val} \in \mathrm{body}(C)\}$
  - $\forall \mathrm{v} \in C_{targets}, C_{rules}(\mathrm{v}) := \{R \in P \mid \mathrm{var}(\mathrm{head}(R)) = \mathrm{v} \wedge \mathrm{head}(R) \in \mathrm{body}(C) \wedge \forall \mathrm{w} \in \mathcal{F}, \forall val, val' \in \mathsf{dom}(\mathrm{w}), \big(\mathrm{w}^{val} \in \mathrm{body}(R) \wedge \mathrm{w}^{val'} \in \mathrm{body}(C)\big) \implies val = val'\}$
    // 4.2) Search for a combination of rules with no conflicting conditions
  - For each $combi \in \times_{\mathrm{v} \in C_{targets}}(C_{rules}(\mathrm{v}))$
    - If $\forall \mathrm{v} \in \mathcal{F}, |\{\mathrm{v}^{val} \in \mathrm{body}(R) \mid val \in \mathsf{dom}(\mathrm{v}) \wedge R \in combi\}| \leq 1$
      - $P'' := P'' \cup \{C\}$
      - **break**
- **OUTPUT:** $P_{\mathcal{O}}(T) \cup C'_{\mathcal{O}}(T) := P \cup P''$.

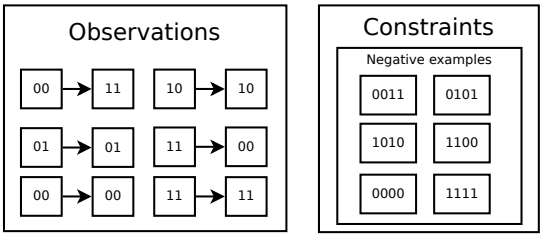


Fig. 8: Preprocessing of the state transitions of Figure 7 (left) into negative examples of the application of constraints.

Theorem 8 (Synchronizer Correctness) *Given any set of transitions T,* **Synchronizer***$(\mathcal{A}, T, \mathcal{F}, \mathcal{T})$ outputs $P_{\mathcal{O}}(T) \cup C'_{\mathcal{O}}(T)$.*

From Theorem 7 and Theorem 8, given a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, it holds that $\mathcal{T}_{syn-c}(Synchronizer(\mathcal{A}, T, \mathcal{F}, \mathcal{T})) = T$, i.e., the algorithm can be used to learn a $\mathcal{CDM}$VLP that reproduce exactly the input transitions whatever the semantics that produced them.

The complexity of the **Synchronizer** is basically a regular call to **GULA** plus a special one to learn constraints and the search for a compatible set of rules in the optimal program which could be blocked by the constraint. Since constraint can have both features and target variables in their body, the complexity of learning constraints with **GULA** is like considering $|\mathcal{F}|+|\mathcal{T}|$ features but only one target value ϵ^1. The detailed complexity of the **Synchronizer** is given in Theorem 9.

Theorem 9 (Synchronizer Complexity) *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ be a set of transitions, let $n := max(|\mathcal{F}|, |\mathcal{T}|)$ and $d := \max(\{|\mathsf{dom}(\mathrm{v})| \in \mathbb{N} \mid \mathrm{v} \in \mathcal{F} \cup \mathcal{T}\})$ and $m := |\mathcal{F}| + |\mathcal{T}|$.*

The worst-case time complexity of **Synchronizer** *when learning from T belongs to $\mathcal{O}((d^{2n}+2nd^{n+1}+nd^{n+2})+(|T|^2+|T|\times(2m^4d^{2m+2}+2m^3d^{m+1}))+(d^{n^n}))$ and its worst-case memory use belongs to $\mathcal{O}((d^{2n}+2nd^{n+1}+nd^{n+2})+(d^{2m}+2md^{m+1}+md^{m+2})+(nd^n))$.*

The **Synchronizer** algorithm does not need any assumption about the semantics of the underlying model but require the full set of observations. However, when dealing with real data, we may only get access to partial observations. That is why we propose in next section a heuristic method to use GULA in such practical cases.

6 Predictions From Partial Observations with Weighted $\mathcal{DM}$VLPs

In this section, we present a heuristic method allowing to use **GULA** to learn from partial observations and predict from unobserved feature states. Previous sections were focusing on theoretical aspects of our method. The two

algorithms presented in Sections 4 and 5 are sound regarding the observations they have been provided as input. Rules of an optimal program provide minimal explanations and can reproduce what is possible over observed transitions. If observation are incomplete, the optimal program will realize a transition to every possible target state from unobserved feature state, i.e. all target atoms are always possible for unobserved feature state. In practice, when observations are partial, to get predictions and explanations from our model on unobserved feature states, we also need to model impossibilities.

Definition 26 (Rule of Impossibility) An *rule of impossibility* of $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ is a $\mathcal{M}$VL rule R such that $R \sqcap s \in \mathcal{S}^{\mathcal{F}}, (s, s') \in T \implies \text{head}(R) \notin s'$.

A rule of impossibility always conflicts with T (Definition 7) but all conflicting rules are not necessarily rules of impossibility. Indeed, considering all feature states matching a rule R, a conflicting rule has at least one feature state that never leads to a target state containing its conclusion; for a rule of impossibility, it is the case of all these matching feature states. The conclusion of a rule of impossibility is never observed in any transition from a feature state it matches, i.e., its body is a condition so that its head is not possible.

Definition 27 (Optimal Program of Impossibility) Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. A $\mathcal{DM}$VLP P *is impossibility-suitable for* T when:

- All rules in P are rules of impossibility of T.
- for all rules of impossibility R of T, there exists $R' \in P$ such that $R' \geq R$.

If in addition, for all $R \in P$, all the $\mathcal{M}$VL rules R' belonging to $\mathcal{DM}$VLP impossibility-suitable for T are such that $R' \geq R$ implies $R \geq R'$ then P is called *impossibility-optimal* and denoted $\overline{P_{\mathcal{O}}(T)}$.

Rules of possibility and impossibility can be weighted according to the observations to form a $\mathcal{WDM}$VLP as given in Definition 28.

Proposition 5 (Uniqueness of Impossibility-Optimal Program) *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. The $\mathcal{DM}$VLP impossibility-optimal for T is unique and denoted $\overline{P_{\mathcal{O}}(T)}$.*

Definition 28 (Weighted $\mathcal{DM}$VLP) A *weighted program* is a set of weighted rules: $\{(w, R) \mid w \in \mathbb{N} \wedge R \text{ is a } \mathcal{DM}\text{VLP rule}\}$. A $\mathcal{WDM}$VLP is a pair of weighted programs (P, P') on the same set of atoms $\mathcal{A}$, and the same feature and target variables $\mathcal{F}$ and $\mathcal{T}$.

Example 21 Let $WP = (P, P')$ be a $\mathcal{WDM}$VLP, as follows.

$P = \{$	$P' = \{$
$(3, a_t^0 \leftarrow b_{t-1}^1)$	$(30, a_t^0 \leftarrow c_{t-1}^1)$
$(15, a_t^1 \leftarrow b_{t-1}^0)$	$(5, a_t^1 \leftarrow c_{t-1}^0)$
$\dots\}$	$\dots\}$

Let $s := \{a_{t-1}^0, b_{t-1}^1, c_{t-1}^1\}$. The rule of possibility $a_t^0 \leftarrow b_{t-1}^1$ matches s, and the rule of impossibility $a_t^0 \leftarrow c_{t-1}^1$ also matches s. The weight of the rule of

impossibility (30) being greater than that of the rule of possibility (3), we can consider that a_t^0 is not likely to appear in a transition from s according to WP.

Using GULA, we can learn both rules of possibility (by using parameter *learning_mode* = "*possibility*") and rules of impossibility (with parameter *learning_mode* = "*impossibility*") from T. In Algorithm 4, **GULA** is used to learn two distinct $\mathcal{DM}$VLPs: a program of possibility and a program of impossibility. The rules of both programs are then weighted by the number of feature states they match to form a weighted $\mathcal{DM}$VLP. This $\mathcal{WDM}$VLP can then be used to make predictions from unseen feature states by confronting the learned rules of possibility and impossibility according to their weights.

Algorithm 4 Learning WDMVLP with GULA

- **INPUT:** a set of atoms $\mathcal{A}'$, a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}'} \times \mathcal{S}^{\mathcal{T}'}$, two sets of variables $\mathcal{F}'$ and $\mathcal{T}'$
- $P := GULA(\mathcal{A}', T, \mathcal{F}', \mathcal{T}', \text{“}possible\text{”})$
- $P' := GULA(\mathcal{A}', T, \mathcal{F}', \mathcal{T}', \text{“}impossible\text{”})$
- $WP := \{(|\{s \in \mathcal{S} \mid (s, s') \in T \wedge R \sqcap s\}|, R) \in \mathbb{N} \times P\}$
- $WP' := \{(|\{s \in \mathcal{S} \mid (s, s') \in T \wedge R \sqcap s\}|, R) \in \mathbb{N} \times P'\}$
- **OUTPUT:** (WP, WP').

Given a feature state s we can predict and explain the likelihood of each target atom by confronting the rules of possibility and impossibility that match s. Here we simply take the rules with the biggest weight from each weighted program. The likelihoods are computed as given in Definition 29.

Definition 29 Let P be a weighted program, $s \in \mathcal{S}^{\mathcal{F}}$ and $\mathrm{v}^{val} \in \mathcal{A}$ with $\mathrm{v} \in \mathcal{T}$. Let $best\_score(P, s, \mathrm{v}^{val}) := (w_{\max}, M)$ where $w_{\max} := \max(\{w \in \mathbb{N} \mid (w, R) \in P\} \cup \{0\})$ and $M := \{R \mid (w_{\max}, R) \in P \wedge \mathrm{head}(R) = \mathrm{v}^{val}, R \sqcap s\}$.

Let $WP = (P, P')$ be a $\mathcal{WDM}$VLP, $s \in \mathcal{S}^{\mathcal{F}}$ and $\mathrm{v}^{val} \in \mathcal{A}$ with $\mathrm{v} \in \mathcal{T}$. Let $best\_possible(WP, s, \mathrm{v}^{val}) := best\_score(P, s, \mathrm{v}^{val})$ and $best\_impossible(WP, s, \mathrm{v}^{val}) := best\_score(P', s, \mathrm{v}^{val})$.

Let $predict(WP, s, \mathrm{v}^{val}) := \frac{1}{2} \times \left(1 + \frac{w - w'}{\max(\{1, w + w'\})}\right)$, with $best\_possible(WP, s, \mathrm{v}^{val}) = (w, M)$ and $best\_impossible(WP, s, \mathrm{v}^{val}) = (w', M')$.

Let $predict\_and\_explain(WP, s, \mathrm{v}^{val}) := (\mathrm{v}^{val}, predict(WP, s, \mathrm{v}^{val}), (w, R), (w', R'))$ where $(w, R) := arb(best\_possible(WP, s, \mathrm{v}^{val}))$ and $(w', R') := arb(best\_impossible(WP, s, \mathrm{v}^{val})))$ with $arb((w'', M)) = (w'', R'')$ where R'' is taken arbitrarily in M if $M \neq \emptyset$, or $R'' := \varnothing$ if $M = \emptyset$.

Intuitively, $predict(WP, s, \mathrm{v}^{val})$ gives a normalized score between 0 and 1 of the likeliness to observe v^{val} after state s, where 0.5 means that we are left inconclusive. In $predict\_and\_explain(WP, s, \mathrm{v}^{val})$, one of the best rules of possibility and rules of impossibility with their respective weights are given as

explanation to the prediction or a weight of 0 and no rule when no rules of possibility (resp. impossibility) match s. The capacity of this heuristic method to predict and explain from unobserved feature states is evaluated in Section 7.

7 Evaluation

In this section, both the scalability, accuracy and explainability of **GULA** are evaluated using Boolean network benchmarks from the biological literature. The scalability of **Synchronizer** is also evaluated (details are given in appendix). All experiments [1] were conducted on one core of an Intel Core i3 (6157U, 2.4 GHz) with 4 Gb of RAM.

In our experiments we use Boolean networks[2] from Boolenet [11] and Pyboolnet [27]. Benchmarks are performed on a wide range of networks. Some of them are small toy examples, while the biggest ones come from biological case study papers like the Boolean model for the control of the mammalian cell cycle [15] or fission yeast [10]. Boolean networks are converted to $\mathcal{DM}$VLP where $\forall v \in \mathcal{V}, \mathsf{dom}(v) = \{0,1\}$. In [11,27] file formats, for each variable, Boolean functions are given in disjunctive normal form (DNF), a disjunction of conjunction clauses that can be considered as a set of Boolean atoms of the form v or $\neg$v. Each clause c of the DNF of a variable v is directly converted into a rule R such that, $\text{head}(R) = \text{v}_t^1$ and $\text{v}'^1_{t-1} \in \text{body}(R) \iff \text{v}' \in c$ and $\text{v}'^0_{t-1} \in \text{body}(R) \iff \neg\text{v}' \in c$. For each such $\mathcal{DM}$VLP the set T of all transitions are generated for the three considered semantics (see Section 3). For each generation, to simulate the cases where Boolean functions are false, each semantics uses a default function that gives $\text{v}^0, \forall \text{v} \in \mathcal{T}$ when no rule $R, \text{v}(\text{head}(R)) = \text{v}$ matches a state. Table 3 provides the number of variables of each benchmark used in our experiments together with the number of transitions under synchronous, asynchronous and general semantics.

7.1 **GULA** Scalability

Figure 9 shows the run time (log scale) of **GULA** (Algorithm 2) and brute force (Algorithm 1) when learning a $\mathcal{WDM}$VLP from Boolean networks (grouped by number of variables) transitions of Table 3. Since we learn $\mathcal{WDM}$VLP the run time corresponds to the sum of two calls to **GULA** (resp. brute force) (possibility and impossibility mode) and the computation of each rule weight (see Algorithm 4). For each benchmark, learning is performed on 10 random

[1] Available at: `https://github.com/Tony-sama/pylfit`. Using command "`python3 evaluations/mlj2020/mlj2020_all.py`" from the repository's `tests` folder, results will be in the `tests/tmp` folder. All experiements were run with the release version 0.2.2 `https://github.com/Tony-sama/pylfit/releases/tag/v0.2.2`.

[2] Original Boolenet Boolean network files: `https://people.kth.se/~dubrova/boolenet.html`. Original PyBoolNet Boolean network files: `https://github.com/hklarner/PyBoolNet/tree/master/PyBoolNet/Repository`.

Benchmark name	Variables	Transitions		
		synchronous	asynchronous	general
n3s1c1a	3	8	14	29
n3s1c1b	3	8	14	31
raf	3	8	13	29
n5s3	5	32	73	213
n6s1c2	6	64	230	1,039
n7s3	7	128	451	2,243
randomnet_n7k3	7	128	394	1,580
xiao_wnt5a	7	128	324	972
arellano_rootstem	9	512	1,940	11,472
davidich_yeast	10	1,024	4,364	38,720
faure_cellcycle	10	1,024	4,273	30,971
fission_yeast	10	1,024	4,157	33,727
budding_yeast	12	4,096	19,975	260,557
n12c5	12	4,096	30,006	1,122,079
tournier_apoptosis	12	4,096	22,530	358,694
dinwoodie_stomatal	13	8,192	53,249	1,521,099
multivalued	13	8,192	49,156	1,049,760
saadatpour_guardcell	13	8,192	53,249	1,521,099

Table 3: Number of variables and total number of transitions under the three semantics of the Boolean networks from Boolenet [11] and PyBoolNet[27] used as benchmark in this experimental section.

subsets of 1%, 5%, 10%, 25%, 50%, 75%, 100% of the whole set of transitions with a time out of 1,000 seconds.

For all benchmarks, we clearly see that **GULA** is more efficient than the trivial brute force enumeration, the difference exponentially increasing whith the number of variables: about 10 times faster with 6 variables and 100 times faster with 9 variables. The brute force method reaches the time out for 10 variables benchmarks and beyond.

For a given number of variables, we observe that for each benchmark the run time increases with the number of transitions until some ratio (for example 50% for 7 variables) at which point more transition can actually speed up the process. More transitions reduce the probability for a rule to be consistent, thus both methods have less rules to check for domination. This tendency is observed on the three semantics. It is important to note that the systems are deterministic with the synchronous semantics and thus the number of transitions in the synchronous case is much lower than for the two other semantics and one may expect better run time. But the quantity of transitions has little impact in fact and most of the run time goes into rule domination check (see Theorem 6). Actually, more input transitions can even imply less learning time for **GULA**. Having more diverse initial states can also allow the sorting of the negatives example to reduce the quantity of specialization made at each step, a freshly revised rule being revised again will not have much non-dominated candidates to generate. For example, for the benchmarks with 13 variables, for some variable values, given 25% of the transitions, the number of stored rules reached several thousands. On the other hand, when given 100% of the

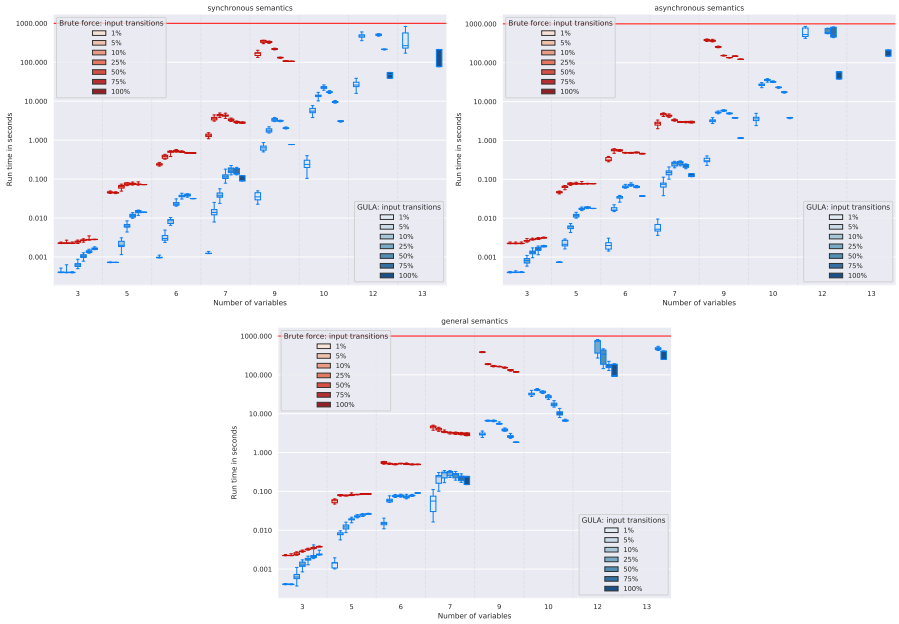


Fig. 9: Run time in seconds (log scale) of two calls to GULA (in blue) and brute force (in red) when learning a WDMVLP from a random set of 1%, 5%, 10%, 25%, 50%, 75%, 100% of the transitions of a Boolean network from Boolenet and PyBoolNet with size varying from 3 to 13 variables. Time out is set at 1,000 seconds and 10 runs where performed for each setting.

transitions, it rarely exceeds hundreds stored rules. Same logic can apply to the faster run time of general semantics with "low" subset of transitions: the total number of transitions being higher, more diversity appears in its subset thus higher chance for the sorting to have effect on reducing the need for least specialization. The rules are simpler for the two other semantics since rules of the form $v_t^{val} \leftarrow v_{t-1}^{val}$ are always consistent and quickly obtained. Such simple rules have great dominance power, reducing the quantity of stored rules and thus checked for domination at each step.

GULA succeeds in learning a $\mathcal{WDMV}$LP from the benchmarks with up to 10 variables for all semantics before the time-out of 1,000 seconds for all considered sub-sets of transitions. Benchmarks from 12 variables need a substantial amount of input transitions to prevent the explosion of consistent rules and thus reaching the time out. For both semantics, the 12 variables benchmarks reached the time out several times when given less than 100% of the transitions. Even if this may seem small compared to the intrinsic complexity of biological systems, ten components are sufficient to capture the dynamic behavior of critical, yet significant, mechanisms like the cell cycle [17].

Compared to our previous algorithm **LF1T** [45], **GULA** is slower in the synchronous deterministic Boolean case (even when learning only $P_{\mathcal{O}}(T)$).

This was expected since it is not specifically dedicated to learning such networks: **GULA** learns both values (0 and 1) of each variable and pre-processes the transitions before learning rules to handle non-determinism. On the other hand, **LF1T** is optimized to only learn rules that make a variable take the value 1 in the next state and assume only one transition from each initial state. furthermore, **LF1T** only handles Boolean values and deterministic transitions while **GULA** can deal with multi-valued variable and any pseudo-idempotent (Theorem 1) semantics transitions.

The current implementation of the algorithm is rather naive and better performances are expected from future optimizations. In particular, the algorithm can be parallelized into as many threads as the number of different rule heads (one thread per target variable value). We are also developing[3] an approximated version of **GULA** that outputs a subset of $P_{\mathcal{O}}(T)$ (resp. $\overline{P_{\mathcal{O}}(T)}$) sufficient to explain T [44]. The complexity of this new algorithm is polynomial, greatly improving the scalability of our approach but to the sacrifice of completeness. Because of space limitations and the ongoing state of this method we could not incorporate this algorithm and its evaluation in this paper.

Learning constraints is obviously more costly than learning regular rules since both feature and target variables can appear in the body, i.e., the number of features becomes $|\mathcal{F}| + |\mathcal{T}|$. Thus by running the **Synchronizer** on the Boolean network benchmark it implies a call to **GULA** with double the number of variables to learn constraints. Under the same experimental settings, the **Synchronizer** reached the time-out of 1,000 seconds on the benchmarks of 7 variables. The contribution regarding $\mathcal{CDM}$VLP being focused on theoretical results, we provided the detailed evaluation of the **Synchronizer** in appendix to save space.

7.2 **GULA** Predictive Power

When addressing biological systems, a major challenge arises: even if the amount of produced data is increasing through the development of high-throughput RNA sequencing, it is still low with regard to all the theoretical contexts.

In this experiment, we thus evaluate the quality of the models learned by **GULA** in their ability to correctly predict possible values for each variable from unseen feature states, i.e., the capacity of the learned model to generalize to unobserved cases. Practically speaking, this ensures the resulting models can provide relevant information about biological experiments that were (or could) not be performed.

For each Boolean network benchmark, we first generate the set of all possible feature states. Those states are then randomly split into two sets: at least 20% will be test feature states and the remaining 80% will be potential

[3] The polynomial approximation of **GULA**, currently named **PRIDE** is also available at: `https://github.com/Tony-sama/pylfit`

training feature states. According to the Boolean formula of the network and a given semantics, all transitions from test feature states are generated to make the test set. All transitions are also computed from the training feature states, but only x% of the transitions are randomly chosen to form the training set with $x \in \{1, 5, 10, 20, 30, ..., 100\}$. Figure 10 illustrates the construct of both training and test sets for a Boolean network of 3 variables.

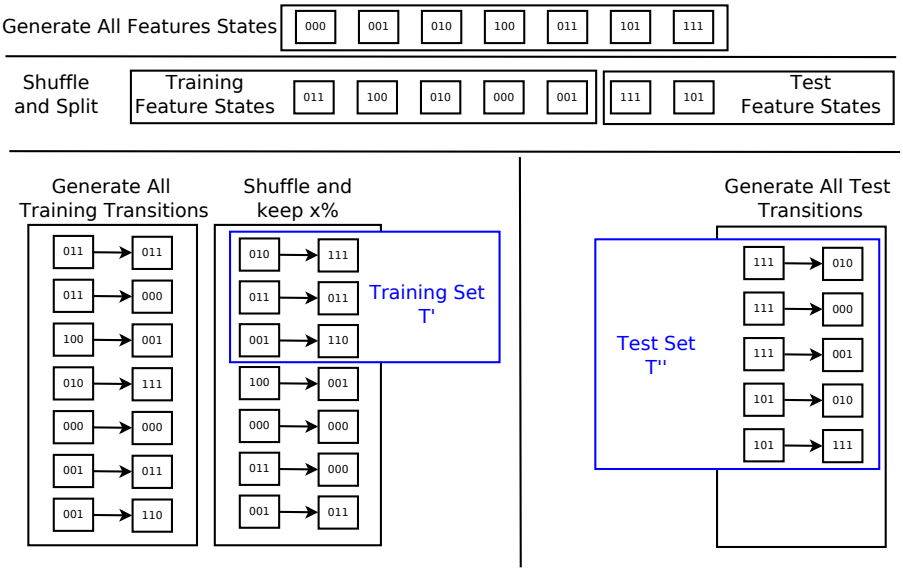


Fig. 10: Experiments settings: data generation, train/test split.

The training set is used as input to learn a WDMVLP using GULA. The learned WDMVLP WP is then used to predict from each feature state s of the test set, the possibility of occurrence of each target atoms v^{val} according to Proposition 29, i.e., $predict(WP, s, \mathrm{v}^{val})$. The forecast probabilities are compared to the observed values of the test set. Let T be the set of all transitions, T' the training set of transitions and T'' the test set of transitions. For all $\mathrm{v}^{val} \in \mathcal{A}|_{\mathcal{T}}$ and $s \in \mathrm{first}(T'')$, we define:

$$actual(\mathrm{v}^{val}, s, T'') = \begin{cases} 1, & \text{if } \exists (s, s') \in T'', \mathrm{v}^{val} \in s' \\ 0, & \text{otherwise} \end{cases}.$$

To evaluate the accuracy of prediction from the learned $\mathcal{WDMV}$LP, WP, over the test set T'' we consider a ratio of precision given by the complement to one of the mean absolute error between its prediction and the actual value:

$$accuracy(WP, T'') = \sum_{s \in \mathrm{first}(T'')} \sum_{\mathrm{v}^{val} \in \mathcal{A}|_{\mathcal{T}}} \frac{1 - |actual(\mathrm{v}^{val}, s, T'') - predict(WP, s, \mathrm{v}^{val})|}{|\mathcal{A}|_{\mathcal{T}}| \times |\mathrm{first}(T'')|}$$

Formally, if T is the whole set of transitions of the Boolean network, this experiment consists in learning an approximation of the pair $(P_{\mathcal{O}}(T), \overline{P_{\mathcal{O}}(T)})$ from the training set $T' \subset T$ and checking both the consistency and realization of the test set $T'' \subset T$, with $\text{first}(T') \cap \text{first}(T'') = \emptyset$, having $|T'| \approx x \times 0.8 \times |T|$ and $|T''| \approx 0.2 \times |T|$, where $x \in \{0.01, 0.05, 0.1, 0.2, 0.3, ..., 1.0\}$.

Example 22 Let T'' be the test set of Figure 10 and WP be the $\mathcal{WDM}$VLP of Example 21. Let $s := (a^1_{t-1}, b^1_{t-1}, c^1_{t-1})$ (111).

- Expected prediction from s according to T'':
  $\{(\mathrm{v}^{val}, actual(\mathrm{v}^{val}, s, T''))\} = \{(a^0_t, 1), (a^1_t, 0), (b^0_t, 1), (b^1_t, 1), (c^0_t, 1), (c^1_t, 1)\}$
- Predictions from s according to WP:
  $\{(\mathrm{v}^{val}, predict(WP, s, \mathrm{v}^{val}))\} = \{(a^0_t, 0.9), (a^1_t, 0.2), (b^0_t, 0.8), (b^1_t, 0.6), (c^0_t, 1.0), (c^1_t, 0.0)\}$
- Accuracy (unique state): $1 - \frac{|1-0.9|+|0-0.2|+|1-0.8|+|1-0.6|+|1-1.0|+|1-0.0|}{|\mathcal{A}|_{\mathcal{T}}|=6} = 0.58$

On state s, the model prediction mean absolute error w.r.t. T'' is 0.42, thus giving an accuracy of 0.58, meaning that in average, 58% of the predictions are correct.

Figures 11a, 11b and 11c show the accuracy of the predicted possible values w.r.t. the ratio of training data going from 1% to 100% with the three considered semantics.

Here, we also consider four trivial baselines that are random predictions and always predicting 0, 0.5 or 1.0, i.e., $\forall s \in \mathcal{S}^{\mathcal{F}}, \forall \mathrm{v}^{val} \in \mathcal{A}|_{\mathcal{T}}$:

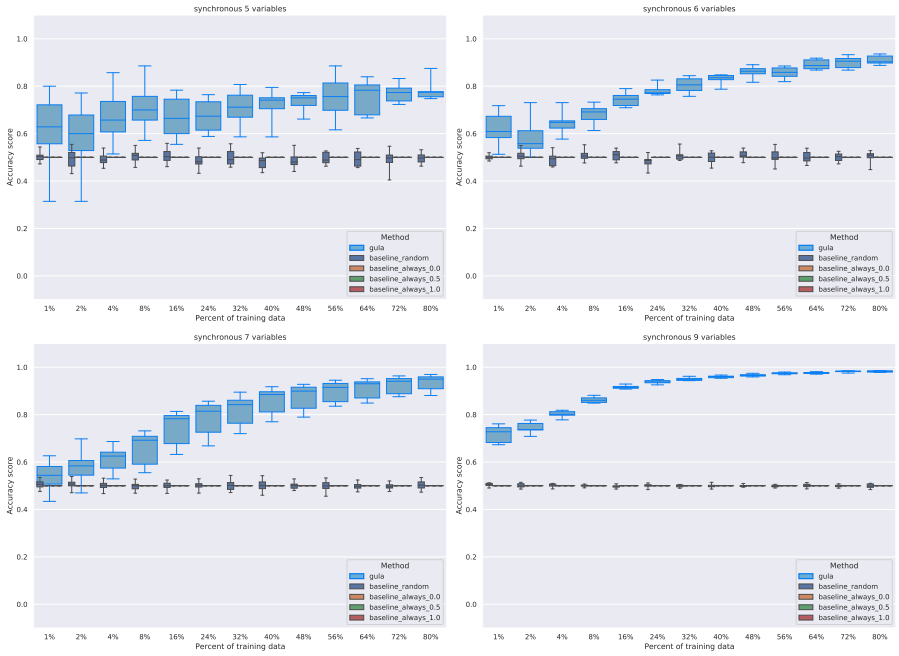
- $baseline\_random(s, \mathrm{v}^{val}) = rand(0.0, 1.0)$
- $baseline\_always\_0.0(s, \mathrm{v}^{val}) = 0.0$
- $baseline\_always\_0.5(s, \mathrm{v}^{val}) = 0.5$
- $baseline\_always\_1.0(s, \mathrm{v}^{val}) = 1.0$

Accuracy score for the random baseline is expected to be around 0.5 for every semantics since the problem is equivalent to a binary classification, i.e., each atom can appear or not. Accuracy score of the three fixed baselines is exactly 0.5 in synchronous case since transitions are deterministic here: only one atom v^{val} is possible (either v^0 or v^1) for each target variable v for each feature state of the test set, i.e., always one of the two must be predicted to 0.0 and the other one to 1.0. For asynchronous and general semantics the transitions are non-deterministic, thus always predicting 0.0 or 1.0 for each target atoms will lead to different accuracy score. Both semantics using previous value as default, it is more likely for each atom to appear in a target state, thus always predicting 1.0 is expected to perform better than 0.5 and always predicting 0.0 is expected to perform worst. That explain why, in Figures 11b and 11c we can observe an accuracy score of 0.6 to 0.8 for always predicting 1.0 and 0.2 to 0.4 for always predicting 0.0.

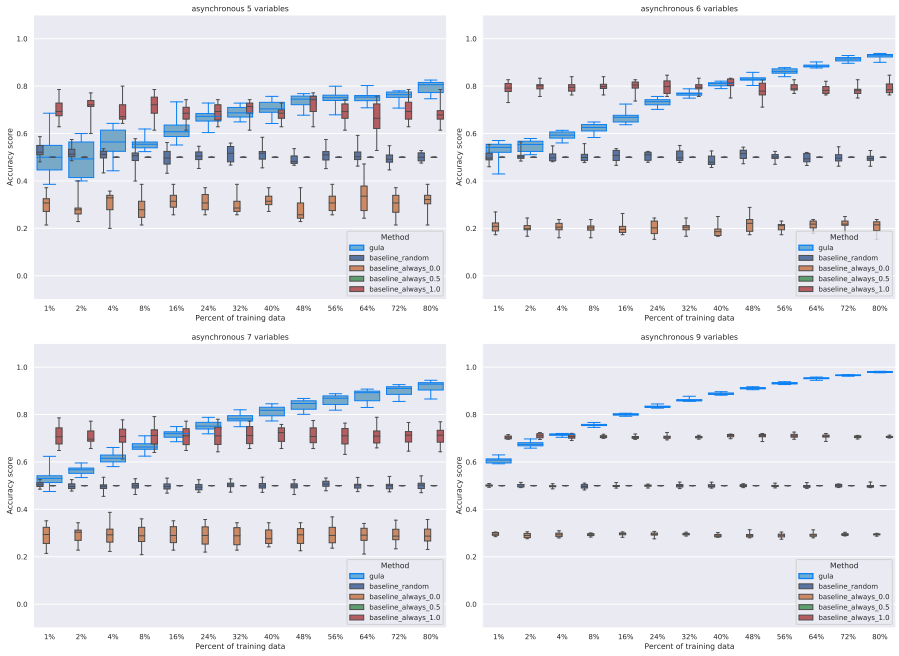
With synchronous semantics transitions, when given only 5% of the possible transitions, **GULA** starts to clearly outperform the baseline on the test set for all benchmarks size. It reaches more than 80% accuracy when given at least 40% of the transitions for benchmarks with 6 variables and only 5%

Fig. 11: Accuracy of the $\mathcal{WDMVLP}$ learned by **GULA** and trivial baselines when predicting possible target atoms from unseen states with different amounts of training data of transitions from Boolean network benchmarks with synchronous, asynchronous and general semantics.

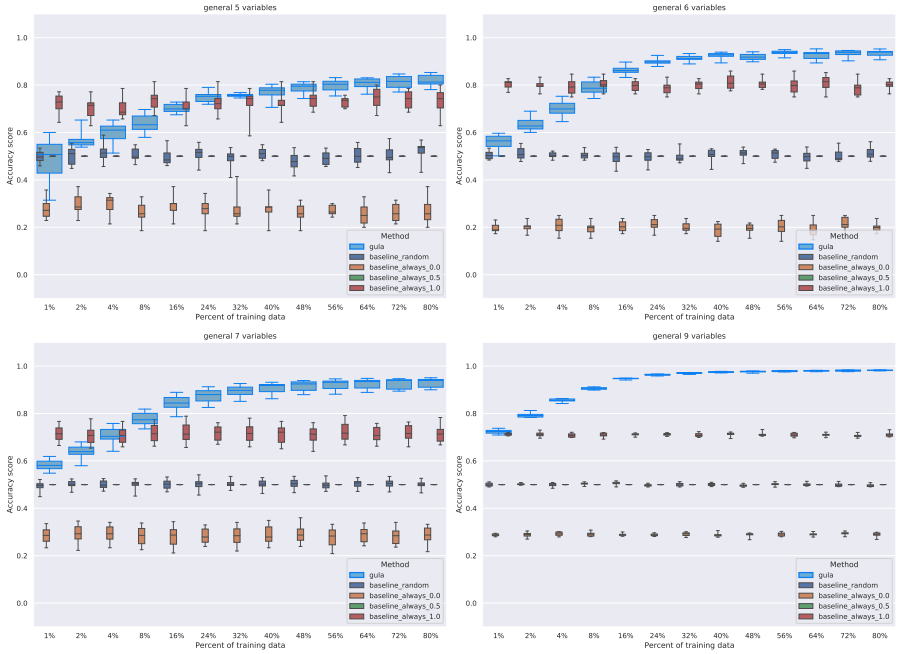
(a) Synchronous semantics.



(b) Asynchronous semantics.



(c) General semantics.



of input transitions is enough to obtain same performance with 9 variables. These results show that the models learned by **GULA** effectively generalise some meaningful behavior from training data over test data in a deterministic context.

For the non-deterministic case of asynchronous and general semantics the performance of **GULA** are similar but the differences with the baselines that always predict 1.0 is smaller. As stated before, since both semantics use previous value as default, it is more likely for each atom to appear in a target state, thus predicting that all atoms are always possible is less risky. Furthermore, the transition being non-deterministic, the way we select the training set (see Figure 10) may lead to have missing transitions from some feature state in the training set, generating false negative example for **GULA** equivalent to noisy data. Still, **GULA** start to outperforms the baseline that always predict 1.0 (and all others) for the two semantics when given more than 50% of the possible transitions as input. The performances of **GULA** also increase when considering more variables, with 9 variables benchmarks 20% of transition is enough to obtain 80% accuracy over unseen test data for asynchronous case and about 2% for general case. Performances are globally similar for the three semantics, showing that our method can handle a bit of noise caused by missing observations.

If one is only interested by prediction accuracy, it is certainly easier to achieve better results using statistical machine learning methods like neural networks or random forest since prediction here is basically a binary classification for each target variables values. In the cases where explainability is of interest, the rules used for the predictions and their weights may be quite simple human readable candidates for explanations (i.e., exhibit dynamic relations between biological interacting components). We consider the evaluation of explanation in following experiment.

7.3 **GULA** Explainability power

In this experiment, we evaluate the quality of the models learned by **GULA** in their ability to correctly explain their predictions. Benchmarks and train/test sets generation is the same as in previous experiment (see Figure 10). The learned model must predict correctly the possibility for each target atom as previously, and also provide a rule that can explain the prediction. When a target atom is possible (resp. impossible), we expect a rule of the optimal program (resp. optimal program of impossibility) to be given as explanation. By computing the Hamming distance between the rules used in the model learned from incomplete observations $(P_{\mathcal{O}}(T'), \overline{P_{\mathcal{O}}(T')})$, and the optimal rules from the full observations $(P_{\mathcal{O}}(T), \overline{P_{\mathcal{O}}(T)})$, we can have an idea of how close we are from the theoretically optimal explanations. For that, for each experiment, we compute the optimal program and the optimal program of impossibility from the set of all transitions (T) before splitting it into train/test sets.

A $\mathcal{WDMM}$VLP is then learned using **GULA** from the training set (T') as in previous experiment. The learned $\mathcal{WDMM}$VLP is then used to predict from each feature state of the test set (T''), the possibility of occurrence of each target atom according to Proposition 29 as well as a rule to explain this prediction. The forecast probabilities and explanations are compared to the observed values of the test set and the rules of the optimal programs. For all $\mathrm{v}^{val} \in \mathcal{A}|_{\mathcal{T}}$ and $s \in \text{first}(T'')$, we define:

$$actual(\mathrm{v}^{val}, s, T'') = \begin{cases} (1, \{R \in P_{\mathcal{O}}(T) \mid \text{head}(R) = \mathrm{v}^{val} \wedge R \sqcap s\}), & \text{if } \exists (s, s') \in T'', \mathrm{v}^{val} \in s' \\ (0, \{R \in \overline{P_{\mathcal{O}}(T)} \mid \text{head}(R) = \mathrm{v}^{val} \wedge R \sqcap s\}), & \text{otherwise} \end{cases} .$$

To compare the forecast rules and the ideal rules, we consider the Hamming distance over their bodies:

$$distance(R, R') = |(\text{body}(R) \cup \text{body}(R')) \setminus (\text{body}(R) \cap \text{body}(R'))| \ .$$

We expect both correct forecast of possibility and explanation, in the sense that an incorrect prediction yields the highest error (1.0) while a good prediction yields an error depending on the quality of the explanation (0.0 when an ideal rule is used). This is summed up in the following error function:

$$error((forecast\_proba, forecast\_rule), (actual\_proba, actual\_rules)) = \begin{cases} 1.0 \text{ if } forecast\_rule = \varnothing \\ 1.0 \text{ if } forecast\_proba = 0.5 \\ 1.0 \text{ if } forecast\_proba > 0.5 \wedge actual\_proba = 0 \\ 1.0 \text{ if } forecast\_proba < 0.5 \wedge actual\_proba = 1 \\ \frac{\min(\{distance(forecast\_rule, R) | R \in actual\_rules\})}{|\mathcal{F}|} \text{ otherwise} \end{cases}$$

This allows to compute an explanation score, combining both accuracy and explanation quality from the learned $\mathcal{WDMM}$VLP, WP, over the test set T'':

$$explanation\_score(WP, T'') = \sum_{s \in \text{first}(T'')} \sum_{\text{v}^{val} \in \mathcal{A}|_{\mathcal{T}}} \frac{1 - |error(predict\_and\_explain(WP, s, \text{v}^{val}), actual(\text{v}^{val}, s, T''))|}{|\mathcal{A}|_{\mathcal{T}}| \times |\text{first}(T'')|}$$

Example 23 Let $\mathcal{F} = \{a_{t-1}, b_{t-1}, c_{t-1}\}$, $\mathcal{T} = \{a_t, b_t, c_t\}$, a complete set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, a train set of transitions $T' \subseteq T$ and a test set of transitions $T'' \subseteq T$ with $T' \cap T'' = \emptyset$ such that:

- $P_{\mathcal{O}}(T) = \{a_t^1 \leftarrow a_{t-1}^1, a_t^1 \leftarrow b_{t-1}^1 \wedge c_{t-1}^1, a_t^1 \leftarrow b_{t-1}^0 \wedge c_{t-1}^0, a_t^0 \leftarrow c_{t-1}^0, ...\}$
- $\overline{P_{\mathcal{O}}(T)} = \{a_t^1 \leftarrow a_{t-1}^0, a_t^1 \leftarrow b_{t-1}^0, a_t^1 \leftarrow c_{t-1}^0, a_t^0 \leftarrow c_{t-1}^1, ...\}$

- Let us suppose that from the test feature state $s := \{a_{t-1}^1, b_{t-1}^1, c_{t-1}^1\}$, the target atom a_t^1 is observed in some transitions from s in T'' thus we expect a probability of 1.0 and a rule from $P_{\mathcal{O}}(T)$ that matches s and produce a_t^1 (any of the blue rules) as explanation:
  - $actual(a_t^1, s, T'') = (a_t^1, 1.0, \{a_t^1 \leftarrow a_{t-1}^1, a_t^1 \leftarrow b_{t-1}^1, c_{t-1}^1\})$
- Let WP be a $\mathcal{WDMM}$VLP learned from T' and we suppose that:
  - $predict\_and\_explain(WP, s, a_t^1) = (a_t^1, 1.0, a_t^1 \leftarrow b_{t-1}^1)$
- The predicted possibility is correct, thus the explanation score will depend on the explanation.
- The explanation $a_t^1 \leftarrow b_{t-1}^1$ has a Hamming distance of 2 with $a_t^1 \leftarrow a_{t-1}^1$ (the conditions on a_{t-1} and b_{t-1} are wrong, the condition on c_{t-1} is correct), thus the error will be $\frac{2}{|\mathcal{F}|} = \frac{2}{3}$.
- The Hamming distance is only of 1 with rule $a_t^1 \leftarrow b_{t-1}^1, c_{t-1}^1$ (the conditions on a_{t-1} and b_{t-1} are correct, the condition on c_{t-1} is wrong), thus the error will be $\frac{1}{|\mathcal{F}|} = \frac{1}{3}$.
- The final score for target a_t^1 is $1 - min(\{\frac{2}{3}, \frac{1}{3}\}) \approx 0.66$

The prediction is correct for target a_t^1 from s, but the explanation $a_t^1 \leftarrow a_{t-1}^1$ is not perfect. Still, 66% of its conditions correspond to an optimal rule ($a_t^1 \leftarrow b_{t-1}^1, c_{t-1}^1$) that can explain this prediction.

- Now let us suppose that from the test feature state $s := \{a_{t-1}^0, b_{t-1}^1, c_{t-1}^0\}$, the target atom a_t^1 is never observed in any transition from s in T''. Thus, we expect a predicted probability of 0.0 and, as an explanation, a rule from $\overline{P_{\mathcal{O}}(T)}$ that matches s and has a_t^1 as conclusion (any of the red rules):
  - $actual(a_t^1, s, T'') = (a_t^1, 0.0, \{a_t^1 \leftarrow a_{t-1}^0, a_t^1 \leftarrow c_{t-1}^0\})$
- Let WP be a $\mathcal{WDMM}$VLP and suppose that:
  - $predict\_and\_explain(WP, s, a_t^1) = (a_t^1, 0.0, a_t^1 \leftarrow \emptyset)$
- The explanation $a_t^1 \leftarrow \emptyset$ has an Hamming distance of 1 when compared with $a_t^1 \leftarrow a_{t-1}^0$ (the condition on a_{t-1} is wrong, the conditions on b_{t-1} and c_{t-1} are correct), thus the error will be $\frac{1}{|\mathcal{F}|} = \frac{1}{3}$.

- We obtain the same Hamming distance of 1 when compared with $a_t^1 \leftarrow c_{t-1}^0$.
- The final score for target a_t^1 from s is $1 - min(\{\frac{1}{3}, \frac{1}{3}\}) \approx 0.66$.

The prediction is correct for target a_t^1 from s, but the explanation $a_t^1 \leftarrow \emptyset$ is not perfect. Still, 66% of its conditions correspond to an optimal rules of impossibility ($a_t^1 \leftarrow a_{t-1}^0$ and $a_t^1 \leftarrow c_{t-1}^0$) that can explain this prediction.

Figures 12a, 12b and 12c show the results of the evolution of the explanation score when learning a $\mathcal{WDMM}$VLP using **GULA** from approximately 1% to 80% of the transitions of a Boolean network. We also use 4 trivial methods as baselines, each having a perfect value prediction, thus their score is only influenced by their explanation. The baselines explanations are trivial and take the form of a random rule, no rules, the most specific rule, the most general rule, i.e., $\forall s \in \text{first}T'', \forall \text{v}^{val} \in \mathcal{A}|_{\mathcal{T}}, perfect\_prediction = actual(\text{v}^{val}, s, T'')$:

- $baseline\_random\_rules(s, \text{v}^{val}) = (perfect\_prediction, \text{v}^{val} \leftarrow body \subseteq s)$
- $baseline\_no\_rules(s, \text{v}^{val}) = (perfect\_prediction, \varnothing)$
- $baseline\_most\_general\_rules(s, \text{v}^{val}) = (perfect\_prediction, \text{v}^{val} \leftarrow \emptyset)$
- $baseline\_most\_specific\_rules(s, \text{v}^{val}) = (perfect\_prediction, \text{v}^{val} \leftarrow s)$

The random baseline is expected to score around 0.5, while the no rule baseline will always have a score of 0.0. The most specific rule baseline will have all conditions of each expected rule, but also unnecessary ones. The most general rules will miss all specific conditions but avoid all unnecessary ones. Since optimal rules rarely use more than half of the total number of variable as conditions (at least for these Benchmarks), the most general rule is expected to have a better score in average compared to most specific. That's why we observe random rule score around 0.4 to 0.5, most specific score around 0.1 to 0.4 and most general score around 0.6 to 0.8 for all semantics considered.

With synchronous semantics transitions, when given only 50% of the possible transitions, **GULA** start to clearly outperform the baselines on the test set for all benchmarks size. It reaches more than 80% accuracy when given at least 25% of the transitions for benchmarks with 6 variables and only 10% of input transitions is enough to obtain same performance with 9 variables. These results show that **GULA**, in a deterministic context, effectively learns rules that are close to the optimal ones even with a partial set of observations, showing its capacity in practice to generalize to unseen data. Such results will help to validate, using the data, models that were previously built and designed by the sole expert knowledge of the biological experts. Meanwhile we cannot rely only on deterministic semantics, as well-known models from the literature (e.g., the switch between the lytic and lysogenic cycles of the lambda phage [51], which is composed of four components in interaction) require non-determinism to be captured efficiently.

For the non-deterministic case of asynchronous and general semantics the performance of **GULA** are similar but more observation are needed to obtain same performances. Like for previous experiments, in those cases we can have missing transitions for some of the observed feature states, leading to false

negative examples extraction in **GULA**. This is more likely to happen with asynchronous semantics, since only one transition will show the change of a specific variable value from a given state while the general semantics will have several subset of change combined in a transitions. It also makes transitions less valuable in quantity of information in the asynchronous case, i.e., only one variable changes its value, starting from the second transition from the same state, all transitions only provide one positive example for the only variable that is changing its value. Still, **GULA** starts to outperform the most general rule baseline (and all others) for the two semantics when given more than 50% of the possible transitions as input. This shows again that our method can handle a bit of noise caused by missing observations also at the explanation level. The performances of **GULA** are similar when considering more variables here, the gain observed in value precision compensating the additional possibility for explanation error introduced by new variables.

It is important to recall that the baselines used here have perfect value prediction while our method also need to predict proper value to have it's explanation evaluated. As stated before, it is certainly easier to achieve better prediction results using statistical machine learning methods. Furthermore, when good prediction model can be built from training data, it can replace our learned model to forecast the value but could be used to improve the output of **GULA**. Indeed, one can use such models to directly generate positive/negative examples of each atom from observed and unseen states that can be given as input to **GULA** in place of the raw observations. It can help to deal with noisy data and improve the diversity of initial state that can speed up and improve the quality of the rules of **GULA** and thus also its approximated version [44]. Actually, as long as feature and target variables are discrete (or can be properly discretized), **GULA** (or its approximated version for big systems) could be used to generate rules that could explain in a more human readable way the behavior of other less explainable models. Such a combination of predictive statistical model and $\mathcal{WDM}$VLP learning study is out of the scope of this paper but will be an interesting application part of our future works. This would not only allow to output relevant predictions w.r.t. dynamical trajectories of biological systems but also help to get a precise understanding of the underlying key interactions between components. Such an approach can also be considered for a broader range of applications. In [38], the authors investigate the promises conveys to provide declarative explanations in classical machine learning by neural networks in the context of automatic recruitment algorithms.

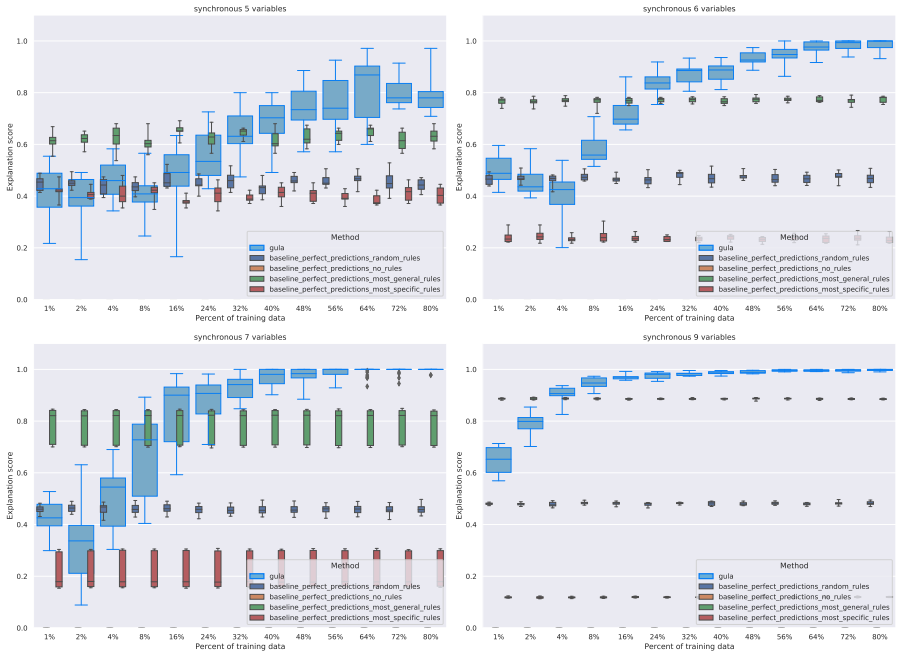
8 Related Work

8.1 Modeling Dynamics

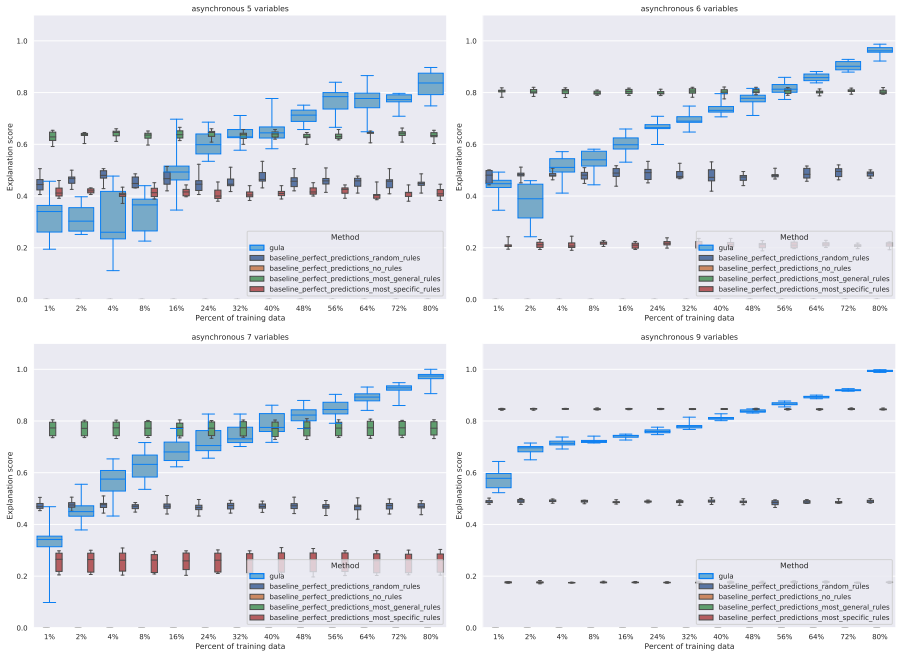
In modeling of dynamical systems, the notion of concurrency is crucial. Historically, two main dynamical semantics have been used in the field of sys-

Fig. 12: Explainability score of the $\mathcal{WDMV}$LP learned by **GULA** and trivial baselines when predicting possible target atoms from unseen states with different amounts of training data of the transitions from Boolean network benchmarks with synchronous, asynchronous and general semantics.

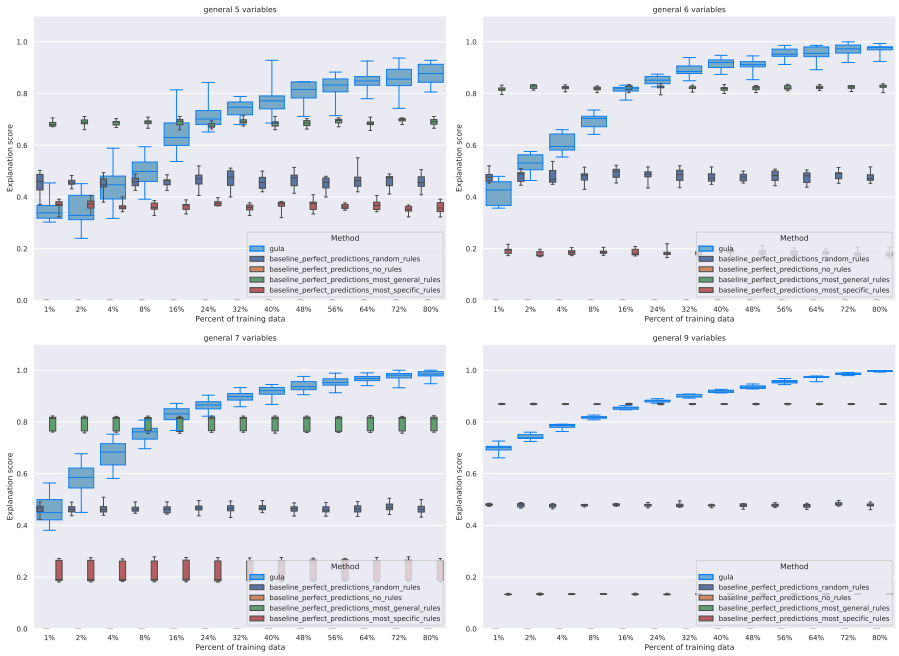
(a) Synchronous semantics.



(b) Asynchronous semantics.



(c) General semantics.



tems biology: synchronous (Boolean networks of Stuart Kauffman [24]) and asynchronous (René Thomas' networks [52]), although other semantics are sometimes proposed or used [14].

The choice of a given semantics has a major impact on the dynamical features of a model: attractors, trap domains, bifurcations, oscillators, etc. The links between modeling frameworks and their update semantics constitute the scope of an increasing number of papers. In [19], the author exhibited the translation from Boolean networks into logic programs and discussed the point attractors in both synchronous and asynchronous semantics. In [37], the authors studied the synchronism-sensitivity of Boolean automata networks with regard to their dynamical behavior (more specifically their asymptotic dynamics). They demonstrate how synchronism impacts the asymptotic behavior by either modifying transient behaviors, making attractors grow or destroying complex attractors. Meanwhile, the respective merits of existing synchronous, asynchronous and generalized semantics for the study of dynamic behaviors has been discussed by Chatain and Paulevé in a series of recent papers. In [7], they introduced a new semantics for Petri nets with read arcs, called the interval semantics. Then they adapted this semantics in the context of Boolean networks [8], and showed in [6] how the interval semantics can capture additional behaviors with regard to the already existing semantics. Their most recent work demonstrates how the most common synchronous and asynchronous se-

mantics in Boolean networks have three major drawbacks that are to be costly for any analysis, to miss some behaviors and to predict spurious behaviors. To overcome these limits, they introduce a new paradigm, called *Most Permissive Boolean Network* which offers the guarantee that no realizable behavior by a qualitative model will be missed [41].

The choice of a relevant semantics appears clearly not only in the recent theoretical works bridging the different frameworks, but also in the features of the software provided to the persons involved in Systems Biology modeling (e.g., the GinSIM tool offers two updating modes, that are fully synchronous and fully asynchronous [36]). Analysis tools offer the modelers the choice of the most appropriate semantics with regard to their own problem.

8.2 Learning Dynamics

In this paper, we proposed new algorithms to learn the dynamics of a system independently of its update semantics, and apply it to learn Boolean networks from the observation of their states transitions. Learning the dynamics of Boolean networks has been considered in bioinformatics in several works [30, 1,39,28,14]. In biological systems, the notion of concurrency is central. When modeling a biological regulatory network, it is necessary to represent the respective evolution of each component of the system. One of the most debated issues with regard to semantics targets the choice of a proper update mode of every component, that is, synchronous (Boolean networks of Stuart Kauffman [24]), or asynchronous (René Thomas' networks [52]), or more complex ones. The differences and common features of different semantics w.r.t. properties of interest (attractors, oscillators, etc.) have thus resulted in an area of research per itself, especially in the field of Boolean networks [37,8,6].

In [14], Fages discussed the differential semantics, stochastic semantics, Boolean semantics, hybrid (discrete and continuous) semantics, Petri net semantics, logic programming semantics and some learning techniques. Rather than focusing on particular semantics, our learning methods are complete algorithms that learn transition rules for any memory-less discrete dynamical systems independently of the update semantics.

As in [39], we can also deal with partial transitions, but will not need to identify or enumerate all possible complete transitions. [40] learns a model as a probability distribution for the next state given the previous state and an action. Here, exactly one dynamic rule fires every time-step, which corresponds to the asynchronous semantics of Definition 15.

In [49], action rules are learned using inductive logic programming but require as input background knowledge. In [3], the authors use logic program as a meta-interpreter to explain the behaviour of a system as stepwise transitions in Petri nets. They produce new possible traces of execution, while our output is an interaction model of the system that aims to explain the observed behavior. In practice, our learned programs can also be used to predict unobserved behavior using some heuristics as shown in the experiments of Section 7.

In [26], Klarner *et al.* provide an optimization-based method for computing model reduction by exploiting the prime implicant graph of the Boolean network. This graph is similar to the rules of $P_{\mathcal{O}}(T)$ that can be learned by **GULA**. But while [26] requires an existing model to work, we are able to learn this model from observations.

In [28], Lähdesmäki *et al.* propose algorithms to infer the truth table of Boolean functions of gene regulatory network from gene expression data. Each positive (resp. negative) example represents a variable configuration that makes a Boolean function true (resp. false). The logic programs learned by **GULA** are a generalization of those truth tables.

8.3 Inductive Logic Programming

From the inductive logic programming point of view, **GULA** performs a general to specific search, also called top-down approach. Algorithmically, **GULA** shares similarities with Progol [33,34] or Aleph [50], two state-of-the-art ILP top-down approaches. Progol combines inverse entailment with general-to-specific search through a refinement graph. **GULA** is limited to propositional logic while those two methods handle first order predicates. Learning the equivalent of $\mathcal{DM}$VLP rules should be possible using Progol or Aleph assuming some proper encoding. But both methods would only learn enough rules to explain the positive examples, whereas **GULA** outputs all optimal rules that can explain these examples. The completeness of the output program is critical when learning constraint of a $\mathcal{CDM}$VLP to guarantee the exact reproduction of the observed transitions. Thus, nor Progol or Aleph can replace **GULA** in the Synchronizer algorithm to learn the optimal $\mathcal{CDM}$VLP. But the completeness of the search of **GULA** comes with a higher complexity cost w.r.t. Progol and Aleph. The search of Progol and Aleph is guided by positives examples. Indeed, given a positive example, Progol performs an admissible A*-like search, guided by compression, over clauses which subsume the most specific clause (corresponding to the example). The search of **GULA** is guided by negative examples. It can also be seen as an A*-like search but for all minimal clauses that subsume none of the most specific clauses corresponding to the negative examples.

[12,13] propose the Apperception Engine, a system able to learn programs from a sequence of state transitions. The first difference is that our approach is limited to propositional atoms while first order logic is considered in this approach. Furthermore, the Aperception Engine can predict the future, retrodict the past, and impute missing intermediate values, while we only consider rules to explain what can happen in a next state. But our input can represent transitions from multiple trajectories, while they consider a single trajectory and thus our setting can be considered as a generalized apperception task in the propositional case. Another major difference is that they only consider deterministic inputs while we also capture non-deterministic behaviors. Given the same kind of single trajectory and a $\mathcal{DM}$VLP (or $\mathcal{CDM}$VLP), it should

be possible to produce candidates past states or to try to fill in missing values. But in practice that would suppose to have many other transitions to build such $\mathcal{DM}$VLP using **GULA** while the Aperception Engine can perform the task with only the given single trajectory. This system can also produce a set of constraints as well as rules. The constraints perform double duty: on the one hand, they restrict the sets of atoms that can be true at same time; on the other hand, they ensure what they call the frame axiom: each atom remains true at the next time-step unless it is overridden by a new fact which is incompatible with it. The constraints of $\mathcal{CDM}$VLP can prevent some combinations of atoms to appear, but only in next states, while in [12,13], constraints can prevent some states to exist anywhere in the sequence, and ensure the conservation of atoms. From Theorem 7, the conservation can also be reproduced by $\mathcal{CDM}$VLP by the right combination of optimal rules and constraints.

In [29] the authors propose a general framework named ILASP for learning answer set programs. ILASP is able to learn choice rules, constraints and preferences over answer sets. Our problem settings is related to what is called "context-dependant" tasks in ILASP. Our input can be straightforwardly represented using ILASP when variables are Boolean, but the learned program does not respect our notion of optimality, and thus our learning goals differ, i.e., we guarantee to miss no potential dynamical influence. Indeed, ILASP minimizes a program as a whole, i.e., the sum of the length of all rules and constraints; in contrast, we aim to minimize each rule and constraint individually and expect to find as many of them in practice and all of them in theory to ensure good properties regarding dynamical semantics.

[23] proposes an incremental method to learn and revise event-based knowledge in the form of Event Calculus programs using XHAIL [42], a system that jointly abduce ground atoms and induce first-order normal logic programs. XHAIL needs to be provided with a set of mode declarations to limit the search space of possible induced rules, while our method do not require background knowledge. Still it is possible to exploit background knowledge with **GULA**: for example one could add heuristic inside the algorithm to discard rules with "too many" conditions; influences among variables, if known, could also be exploited to reduce possible bodies. Finally, XHAIL does not model constraints, thus is not able to prevent some combinations of atoms to appear in transitions, which can be achieve using our **Synchronizer**.

9 Conclusions

While modeling a dynamical system, the choice of a proper semantics is critical for the relevance of the subsequent analysis of the dynamics. The works presented in this paper aim to widen the possibilities offered to a system designer in the learning phase. Until now, the systems that the LFIT framework handles were restricted to synchronous deterministic dynamics. However, many other dynamics exist in the field of logical modeling, in particular the asynchronous and generalized semantics which are of deep interest to model biological sys-

tems. In this paper, we proposed a modeling of memory-less multi-valued dynamic systems in the form of annotated logic programs and a first algorithm, **GULA**, that learns optimal programs for a wide range of semantics (see Theorem 1) including notably the asynchronous and generalized semantics. But the semantics need to be assumed to use the learned model, in order to produce predictions for example. Our second proposition is a new approach that makes a decisive step in the full automation of logical learning of models directly from time series, e.g., gene expression measurements along time (whose intrinsic semantics is unknown or even changeable). The **Synchronizer** algorithm that we proposed is able to learn a whole system dynamics, including its semantics, in the form of a single propositional logic program. This logic program explains the behavior of the system in the form of human readable propositional logic rules, as well as, be able to reproduce the behavior of the observed system without the need of knowing its semantics. Furthermore, the semantics can be explained, without any previous assumption, in the form of human readable rules inside the logic program.

This provides a precious output when dealing with real-life data coming from, e.g., biology. Typically, time series data capturing protein (i.e., gene) expressions come without any assumption on the most appropriate semantics to capture the relevant dynamical behaviors of the system. The methods introduced in this paper generate a readable view of the relationships between the different biological components at stake. **GULA** can be used when biological collaborators provide partial observations (as shown by our experiments), for example when addressing gene regulatory networks. Meanwhile the **Synchronizer** algorithm is of interest for systems with the full set of observations, e.g., when refining a model that was manually built by experts.

We took care to show the benefits of our approach on several benchmarks. While systems with ten components are able to capture the behavior of complex biological systems, we exhibit that our implementation is scalable to systems up to 10 components on a computer as simple as a single-core computer with a 1000 seconds time-out. Further work will consist in a practical use of our method on open problems coming from systems biology.

An approximate version of the method is a necessity to tackle large systems and is under development [44]. In addition, lack of observations and noise handling is also an issue when working with biological data. Data science methodologies and deep learning techniques can then be good candidates to tackle this challenge. The combination of such techniques to improve our method may be of prime interest to tackle real data.

References

1. Akutsu, T., Kuhara, S., Maruyama, O., Miyano, S.: Identification of genetic networks by strategic gene disruptions and gene overexpressions under a boolean model. Theoretical Computer Science **298**(1), 235–251 (2003)
2. Apt, K.R., Blair, H.A., Walker, A.: Towards a theory of declarative knowledge. Foundations of deductive databases and logic programming p. 89 (1988)

3. Bain, M., Srinivasan, A.: Identification of biological transition systems using meta-interpreted logic programs. Machine Learning **107**(7), 1171–1206 (2018)
4. Blair, H.A., Subrahmanian, V.: Paraconsistent foundations for logic programming. Journal of non-classical logic **5**(2), 45–73 (1988)
5. Blair, H.A., Subrahmanian, V.: Paraconsistent logic programming. Theoretical Computer Science **68**(2), 135 – 154 (1989). DOI http://dx.doi.org/10.1016/0304-3975(89)90126-6. URL `http://www.sciencedirect.com/science/article/pii/0304397589901266`
6. Chatain, T., Haar, S., Kolčák, J., Paulevé, L., Thakkar, A.: Concurrency in boolean networks. Natural Computing **19**(1), 91–109 (2020)
7. Chatain, T., Haar, S., Koutny, M., Schwoon, S.: Non-atomic transition firing in contextual nets. In: International Conference on Applications and Theory of Petri Nets and Concurrency, pp. 117–136. Springer (2015)
8. Chatain, T., Haar, S., Paulevé, L.: Boolean networks: Beyond generalized asynchronicity. In: AUTOMATA 2018. Springer (2018)
9. Cropper, A., Dumančić, S., Muggleton, S.H.: Turning 30: New ideas in inductive logic programming. In: C. Bessiere (ed.) Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20, pp. 4833–4839. International Joint Conferences on Artificial Intelligence Organization (2020). DOI 10.24963/ijcai.2020/673. URL `https://doi.org/10.24963/ijcai.2020/673`. Survey track
10. Davidich, M.I., Bornholdt, S.: Boolean network model predicts cell cycle sequence of fission yeast. PloS one **3**(2), e1672 (2008)
11. Dubrova, E., Teslenko, M.: A SAT-based algorithm for finding attractors in synchronous boolean networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) **8**(5), 1393–1399 (2011)
12. Evans, R., Hernandez-Orallo, J., Welbl, J., Kohli, P., Sergot, M.: Making sense of sensory input. arXiv preprint 1910.02227 (2019)
13. Evans, R., Hernandez-Orallo, J., Welbl, J., Kohli, P., Sergot, M.: Evaluating the apperception engine. arXiv preprint 2007.05367 (2020)
14. Fages, F.: Artificial intelligence in biological modelling. In: A Guided Tour of Artificial Intelligence Research, pp. 265–302. Springer (2020)
15. Fauré, A., Naldi, A., Chaouiya, C., Thieffry, D.: Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle. Bioinformatics **22**(14), e124–e131 (2006)
16. Fitting, M.: Bilattices and the semantics of logic programming. The Journal of Logic Programming **11**(2), 91 – 116 (1991). DOI http://dx.doi.org/10.1016/0743-1066(91)90014-G. URL `http://www.sciencedirect.com/science/article/pii/074310669190014G`
17. Gibart., L., Bernot., G., Collavizza., H., Comet., J.: Totembionet enrichment methodology: Application to the qualitative regulatory network of the cell metabolism. In: Proceedings of the 14th International Joint Conference on Biomedical Engineering Systems and Technologies - BIOINFORMATICS,, pp. 85–92. INSTICC, SciTePress (2021). DOI 10.5220/0010186200850092
18. Ginsberg, M.L.: Multivalued logics: A uniform approach to reasoning in artificial intelligence. Computational intelligence **4**(3), 265–316 (1988)
19. Inoue, K.: Logic programming for boolean networks. In: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, *IJCAI'11*, vol. 2, p. 924–930. AAAI Press (2011)
20. Inoue, K., Ribeiro, T., Sakama, C.: Learning from interpretation transition. Machine Learning **94**(1), 51–79 (2014)
21. Inoue, K., Sakama, C.: Oscillating behavior of logic programs. In: Correct Reasoning, pp. 345–362. Springer (2012)
22. Kaplan, S., Bren, A., Dekel, E., Alon, U.: The incoherent feed-forward loop can generate non-monotonic input functions for genes. Molecular systems biology **4**(1), 203 (2008)
23. Katzouris, N., Artikis, A., Paliouras, G.: Incremental learning of event definitions with inductive logic programming. Machine Learning **100**(2-3), 555–585 (2015)
24. Kauffman, S.A.: Metabolic stability and epigenesis in randomly constructed genetic nets. Journal of theoretical biology **22**(3), 437–467 (1969)

25. Kifer, M., Subrahmanian, V.: Theory of generalized annotated logic programming and its applications. Journal of Logic Programming **12**(4), 335–367 (1992)
26. Klarner, H., Bockmayr, A., Siebert, H.: Computing symbolic steady states of boolean networks. In: Cellular Automata, pp. 561–570. Springer (2014)
27. Klarner, H., Streck, A., Siebert, H.: PyBoolNet: a python package for the generation, analysis and visualization of boolean networks. Bioinformatics **33**(5), 770–772 (2016). DOI 10.1093/bioinformatics/btw682. URL `https://doi.org/10.1093/bioinformatics/btw682`
28. Lähdesmäki, H., Shmulevich, I., Yli-Harja, O.: On learning gene regulatory networks under the boolean network model. Machine Learning **52**(1-2), 147–167 (2003)
29. Law, M., Russo, A., Broda, K.: Iterative learning of answer set programs from context dependent examples. Theory and Practice of Logic Programming **16**(5-6), 834–848 (2016). DOI 10.1017/S1471068416000351
30. Liang, S., Fuhrman, S., Somogyi, R.: Reveal, a general reverse engineering algorithm for inference of genetic network architectures. In: Proceedings of the 3rd Pacific Symposium on Biocomputing, pp. 18–29 (1998)
31. Martınez, D., Alenya, G., Torras, C., Ribeiro, T., Inoue, K.: Learning relational dynamics of stochastic domains for planning. In: Proceedings of the 26th International Conference on Automated Planning and Scheduling (2016)
32. Martínez Martínez, D., Ribeiro, T., Inoue, K., Alenyà Ribas, G., Torras, C.: Learning probabilistic action models from interpretation transitions. In: Proceedings of the Technical Communications of the 31st International Conference on Logic Programming (ICLP 2015), pp. 1–14 (2015)
33. Muggleton, S.: Inverse entailment and progol. New generation computing **13**(3-4), 245–286 (1995)
34. Muggleton, S.: Learning from positive data. In: International Conference on Inductive Logic Programming, pp. 358–376. Springer (1996)
35. Muggleton, S., De Raedt, L., Poole, D., Bratko, I., Flach, P., Inoue, K., Srinivasan, A.: Ilp turns 20. Machine learning **86**(1), 3–23 (2012)
36. Naldi, A., Hernandez, C., Abou-Jaoudé, W., Monteiro, P.T., Chaouiya, C., Thieffry, D.: Logical modeling and analysis of cellular regulatory networks with ginsim 3.0. Frontiers in physiology **9**, 646 (2018)
37. Noual, M., Sené, S.: Synchronism versus asynchronism in monotonic boolean automata networks. Natural Computing **17**(2), 393–402 (2018)
38. Ortega, A., Fierrez, J., Morales, A., Wang, Z., Ribeiro, T.: Symbolic ai for xai: Evaluating lfit inductive programming for fair and explainable automatic recruitment. target **1**(v1), 1 (2020)
39. Pal, R., Ivanov, I., Datta, A., Bittner, M.L., Dougherty, E.R.: Generating boolean networks with a prescribed attractor structure. Bioinformatics **21**(21), 4021–4025 (2005)
40. Pasula, H.M., Zettlemoyer, L.S., Kaelbling, L.P.: Learning symbolic models of stochastic domains. Journal of Artificial Intelligence Research **29**, 309–352 (2007)
41. Paulevé, L., Kolčák, J., Chatain, T., Haar, S.: Reconciling qualitative, abstract, and scalable modeling of biological networks. bioRxiv (2020)
42. Ray, O.: Nonmonotonic abductive inductive learning. Journal of Applied Logic **7**(3), 329–340 (2009)
43. Ribeiro, T., Folschette, M., Magnin, M., Roux, O., Inoue, K.: Learning dynamics with synchronous, asynchronous and general semantics. In: International Conference on Inductive Logic Programming, pp. 118–140. Springer (2018)
44. Ribeiro, T., Folschette, M., Trilling, L., Glade, N., Inoue, K., Magnin, M., Roux, O.: Les enjeux de l'inférence de modèles dynamiques des systèmes biologiques à partir de séries temporelles. In: C. Lhoussaine, E. Remy (eds.) Approches symboliques de la modélisation et de l'analyse des systèmes biologiques. ISTE Editions (2020). In edition.
45. Ribeiro, T., Inoue, K.: Learning prime implicant conditions from interpretation transition. In: Inductive Logic Programming, pp. 108–125. Springer (2015)
46. Ribeiro, T., Magnin, M., Inoue, K., Sakama, C.: Learning delayed influences of biological systems. Frontiers in Bioengineering and Biotechnology **2**, 81 (2015)
47. Ribeiro, T., Magnin, M., Inoue, K., Sakama, C.: Learning multi-valued biological models with delayed influence from time-series observations. In: 2015 IEEE 14th International

Conference on Machine Learning and Applications (ICMLA), pp. 25–31 (2015). DOI 10.1109/ICMLA.2015.19
48. Ribeiro, T., Tourret, S., Folschette, M., Magnin, M., Borzacchiello, D., Chinesta, F., Roux, O., Inoue, K.: Inductive learning from state transitions over continuous domains. In: N. Lachiche, C. Vrain (eds.) Inductive Logic Programming, pp. 124–139. Springer International Publishing, Cham (2018)
49. Schüller, P., Benz, M.: Best-effort inductive logic programming via fine-grained cost-based hypothesis generation. Machine Learning **107**(7), 1141–1169 (2018)
50. Srinivasan, A.: The aleph manual (2001)
51. Thieffry, D., Thomas, R.: Dynamical behaviour of biological regulatory networks—ii. immunity control in bacteriophage lambda. Bulletin of mathematical biology **57**(2), 277–297 (1995)
52. Thomas, R.: Regulatory networks seen as asynchronous automata: a logical description. Journal of Theoretical Biology **153**(1), 1–23 (1991)
53. Van Emden, M.H.: Quantitative deduction and its fixpoint theory. The Journal of Logic Programming **3**(1), 37–53 (1986)
54. Van Emden, M.H., Kowalski, R.A.: The semantics of predicate logic as a programming language. Journal of the ACM (JACM) **23**(4), 733–742 (1976)

A Appendix: Proofs of Section 2

Lemma 1: Double Domination Is Equality *Let R_1, R_2 be two $\mathcal{M}$VL rules. If $R_2 \geq R_1$ and $R_1 \geq R_2$ then $R_1 = R_2$.*
Proof. Let R_1, R_2 be two $\mathcal{M}$VL rules such that $R_2 \geq R_1$ and $R_1 \geq R_2$. Then $\text{head}(R_1) = \text{head}(R_2)$ and $\text{body}(R_1) \subseteq \text{body}(R_2)$ and $\text{body}(R_2) \subseteq \text{body}(R_1)$, hence $\text{body}(R_1) \subseteq \text{body}(R_2) \subseteq \text{body}(R_1)$ thus $\text{body}(R_1) = \text{body}(R_2)$ and $R_1 = R_2$. □

Proposition 1: Uniqueness of Optimal Program *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. The $\mathcal{M}$VLP optimal for T is unique and denoted $P_{\mathcal{O}}(T)$.*
Proof. Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. Assume the existence of two distinct $\mathcal{M}$VLPs 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 $\mathcal{M}$VL rule R such that $R \in P_{\mathcal{O}_1}(T)$ and $R \notin P_{\mathcal{O}_2}(T)$. By the definition of a suitable program, R is not conflicting with T and there exists a $\mathcal{M}$VL rule $R_2 \in P_{\mathcal{O}_2}(T)$, such that $R_2 \geq R$. Using the same definition, there exists $R_1 \in P_{\mathcal{O}_1}(T)$ such that $R_1 \geq R_2$ since R_2 is not conflicting with T. Thus $R_1 \geq R$ and by the definition of an optimal program $R \geq R_1$. By Lemma 1, $R_1 = R$, thus $R_2 \geq R$ and $R \geq R_2$ hence $R_2 = R$, a contradiction. □

B Appendix: Proofs of Section 3

Theorem 1: Pseudo-idempotent Semantics and Optimal $\mathcal{DM}$VLP *Let DS be a dynamical semantics.*
If, for all P a $\mathcal{DM}$VLP, there exists $\mathsf{pick} \in (\mathcal{S}^{\mathcal{F}} \times \wp(\mathcal{A}|_{\mathcal{T}}) \to \wp(\mathcal{S}^{\mathcal{T}}) \setminus \{\emptyset\})$ so that:

(1) $\forall D \subseteq \mathcal{A}|_{\mathcal{T}}, \mathsf{pick}(s, \bigcup_{s' \in \mathsf{pick}(s,D)} s') = \mathsf{pick}(s, D)$, *and*
(2) $\forall s \in \mathcal{S}^{\mathcal{F}}, \big(DS(P)\big)(s) = \mathsf{pick}(s, \mathsf{Conclusions}(s, P))$,

then, for all P a $\mathcal{DM}$VLP, $DS(P_{\mathcal{O}}(DS(P)))) = DS(P)$.
Proof. Let DS be a dynamical semantics, P a $\mathcal{DM}$VLP, pick a function from $\mathcal{S}^{\mathcal{F}} \times \wp(\mathcal{A}_{\mathcal{T}})$ to $\wp(\mathcal{S}^{\mathcal{T}}) \setminus \{\emptyset\}$ with the properties described in (1) and (2).

In this proof, we use the following equivalent notations, for all $(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$: $(s, s') \in DS(P) \iff s' \in \big(DS(P)\big)(s)$.

By Definition 10, $\text{first}(DS(P)) = \mathcal{S}^{\mathcal{F}}$ $(*)$.

By Definition 9, $P_{\mathcal{O}}(DS(P))$ realizes $DS(P)$. Therefore, according to Definition 5, for all (s, s') in $DS(P)$ and v^{val} in s', because $\mathrm{v} \in \mathcal{T}$, there exists R in $P_{\mathcal{O}}(DS(P))$ so that $\text{var}(\text{head}(R)) = \mathrm{v} \wedge R \sqcap s \wedge \text{head}(R) \in s'$. Because of Definition 3, a discrete state cannot contain two different atoms on the same variable: from $\text{var}(\text{head}(R)) = \mathrm{v} \wedge \mathrm{v}^{val} \in s' \wedge \text{head}(R) \in s'$, it comes: $\text{head}(R) = \mathrm{v}^{val}$. Moreover, by definition of $\mathsf{Conclusions}$, because $R \in P \wedge R \sqcap s$, we have: $\mathrm{v}^{val} \in \mathsf{Conclusions}(s, P_{\mathcal{O}}(DS(P)))$. By generalizing on all v^{val}, it comes: $s' \subseteq \mathsf{Conclusions}(s, P_{\mathcal{O}}(DS(P)))$. By generalizing on all s', it comes: $\forall s \in \mathcal{S}^{\mathcal{F}}, \bigcup_{s' \in (DS(P))(s)} s' \subseteq \mathsf{Conclusions}(s, P_{\mathcal{O}}(DS(P)))$ $(\dagger)$.

By Definition 9, $P_{\mathcal{O}}(DS(P))$ is also consistent with $DS(P)$. Therefore, according to Definition 7: $\forall R \in P_{\mathcal{O}}(DS(P)), \forall s \in \text{first}(DS(P)), R \sqcap s \implies \exists s' \in \big(DS(P)\big)(s), \text{head}(R) \in s'$. From $(*)$, $\text{first}(DS(P)) = \mathcal{S}^{\mathcal{F}}$, thus $\forall s \in \mathcal{S}^{\mathcal{F}}, \forall \mathrm{v}^{val} \in \mathsf{Conclusions}(s, P_{\mathcal{O}}(DS(P))), \exists s' \in DS(P)(s), \mathrm{v}^{val} \in s'$. Thus: $\forall s \in \mathcal{S}^{\mathcal{F}}, \mathsf{Conclusions}(s, P_{\mathcal{O}}(DS(P))) \subseteq \bigcup_{s' \in (DS(P))(s)} s'$ $(\S)$.

From $(\dagger)$ and $(\S)$: $\forall s \in \mathcal{S}^{\mathcal{F}}, \mathsf{Conclusions}(s, P_{\mathcal{O}}(DS(P))) = \bigcup_{s' \in (DS(P))(s)} s'$ $(\star)$.

From $(\star)$ and (2): $\forall s \in \mathcal{S}^{\mathcal{F}}, \mathsf{Conclusions}(s, P_{\mathcal{O}}(DS(P))) = \bigcup_{s' \in \mathsf{pick}(s, \mathsf{Conclusions}(s,P))} s'$ $(\Diamond)$.

Let s in $\mathcal{S}^{\mathcal{F}}$.

– From (2): $\big(DS(P_{\mathcal{O}}(DS(P)))\big)(s) = \mathsf{pick}(s, \mathsf{Conclusions}(s, P_{\mathcal{O}}(DS(P))))$.

- From ($\Diamond$): $\big(DS(P_{\mathcal{O}}(DS(P)))\big)(s) = \mathsf{pick}(s, \bigcup_{s' \in \mathsf{pick}(s, \mathsf{Conclusions}(s,P))} s')$
- From (1): $\big(DS(P_{\mathcal{O}}(DS(P)))\big)(s) = \mathsf{pick}(s, \mathsf{Conclusions}(s, P))$
- From (2): $\big(DS(P_{\mathcal{O}}(DS(P)))\big)(s) = \big(DS(P)\big)(s)$.

Thus: $\forall s \in \mathcal{S}^{\mathcal{F}}, \big(DS(P_{\mathcal{O}}(DS(P)))\big)(s) = \big(DS(P)\big)(s)$, QED. □

Theorem 2 Semantics-Free Correctness *Let P be a $\mathcal{DM}$*VLP.

- $\mathcal{T}_{syn}(P) = \mathcal{T}_{syn}(P_{\mathcal{O}}(\mathcal{T}_{syn}(P)))$,
- $\mathcal{T}_{asyn}(P) = \mathcal{T}_{asyn}(P_{\mathcal{O}}(\mathcal{T}_{asyn}(P)))$,
- $\mathcal{T}_{gen}(P) = \mathcal{T}_{gen}(P_{\mathcal{O}}(\mathcal{T}_{gen}(P)))$.

Proof. Let $d \in (\mathcal{S}^{\mathcal{F}} \times \wp(\mathcal{T}) \to \wp(\mathcal{A}_{\mathcal{T}}))$, so that $\forall s \in \mathcal{S}^{\mathcal{F}}, \forall W \subseteq \mathcal{T}, W \subseteq \text{var}(d(s,W)) \wedge d(s,\emptyset) \subseteq d(s,W)$.

Let p be a function from $\mathcal{S}^{\mathcal{F}} \times \wp(\mathcal{A}_{\mathcal{T}})$ to $\wp(\mathcal{S}^{\mathcal{T}}) \setminus \{\emptyset\}$ so that $\forall s \in \mathcal{S}^{\mathcal{F}}, \forall D \subseteq \mathcal{A}_{\mathcal{T}}, p(s,D) = \{s' \in \mathcal{S}^{\mathcal{T}} \mid s' \subseteq D \cup d(s, \mathcal{T} \setminus \text{var}(D))\}$. Since $\mathcal{T} \setminus \text{var}(D) \subseteq \text{var}(d(s,W)), \emptyset \notin p(s,D)$. Thus from Definition 14, $\forall s \in \mathcal{S}^{\mathcal{F}}, \mathcal{T}_{syn}(P)(s) = p(s, \mathsf{Conclusions}(s,P))$ (property 1).

Since $\forall W \subseteq \mathcal{T}, d(s,\emptyset) \subseteq d(s,W)$, $\forall D \subseteq \mathcal{A}_{\mathcal{T}}, d(s,\emptyset) \subseteq D \cup d(s, \mathcal{T} \setminus \text{var}(D))$, thus $d(s,\emptyset) \subseteq \bigcup_{s' \in p(s,D)} s'$ (property 2).

Moreover, $\forall D \subseteq \mathcal{A}_{\mathcal{T}}$, let $D' := \bigcup_{s' \in p(s,D)} s'$. Straightforwardly: $D' = D \cup d(s, \mathcal{T} \setminus \text{var}(D))$ because we can always create a state with any atom in $D \cup d(s, \mathcal{T} \setminus \text{var}(D))$, thus all atoms of this set are in D', and conversely (property 3). $p(s,D') = \{s' \in \mathcal{S}^{\mathcal{T}} \mid s' \subseteq D' \cup d(s, \mathcal{T} \setminus \text{var}(D'))\}$ by definition of p. $p(s,D') = \{s' \in \mathcal{S}^{\mathcal{T}} \mid s' \subseteq D' \cup d(s,\emptyset)\}$ since $\text{var}(D') = \mathcal{T}$ by definition of D' and p. $p(s,D') = \{s' \in \mathcal{S}^{\mathcal{T}} \mid s' \subseteq D'\}$ from property 2. $p(s,D') = \{s' \in \mathcal{S}^{\mathcal{T}} \mid s' \in D \cup d(s, \mathcal{T} \setminus \text{var}(D))\} = p(s,D)$ from property 3. Therefore p respects (1). Since $\mathcal{T}_{syn}(P) = p(s, \mathsf{Conclusions}(s,P))$, p also respects (2). Thus, $\mathcal{T}_{syn}(P) = \mathcal{T}_{syn}(P_{\mathcal{O}}(\mathcal{T}_{syn}(P)))$ according to Theorem 1.

By definition of $\mathcal{T}_{gen}$: $\forall s \in \mathcal{S}^{\mathcal{F}}, (\mathcal{T}_{gen}(P))(s) = \{s' \in \mathcal{S}^{\mathcal{T}} \mid s' \subseteq \mathsf{Conclusions}(s,P) \cup d(s, T \setminus \text{var}(\mathsf{Conclusions}(s,P)))\}$ with $\mathsf{sp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(s) \subseteq d(s,\emptyset)$. Thus, the same proof gives $\mathcal{T}_{gen}(P) = \mathcal{T}_{gen}(P_{\mathcal{O}}(\mathcal{T}_{gen}(P)))$ according to Theorem 1.

[Let us show that: $\mathcal{T}_{asyn}(P) = \mathcal{T}_{asyn}(P_{\mathcal{O}}(\mathcal{T}_{asyn}(P)))$.] Let p be a function from $\mathcal{S}^{\mathcal{F}} \times \wp(\mathcal{A}_{\mathcal{T}})$ to $\wp(\mathcal{S}^{\mathcal{T}}) \setminus \{\emptyset\}$ so that $\forall s \in \mathcal{S}^{\mathcal{F}}, \forall D \subseteq \mathcal{A}_{\mathcal{T}}$:

$$p(s,D) = \{s' \in \mathcal{S}^{\mathcal{T}} \mid s' \subseteq D \cup d(s, \mathcal{T} \setminus \text{var}(D)) \wedge \big(|s' \setminus \mathsf{sp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(s)| - |\mathcal{T} \setminus \overline{\mathcal{T}}| = 1 \vee (D \cup d(s, \mathcal{T} \setminus \text{var}(D)))_{\overline{\mathcal{T}}} = \mathsf{sp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(s)\big)\}$$

where $\mathcal{A}_{\mathcal{T}}$ and $D_{\overline{\mathcal{T}}}$ are restriction notations from Definition 12. From Definition 15, we have: $\mathcal{T}_{asyn}P = p(s, \mathsf{Conclusions}(s,P))$.

[Let us show that: $\forall D \subseteq \mathcal{A}_{\mathcal{T}}, p(s, \bigcup_{s' \in p(s,D)} s') = p(s,D)$.] Let D in $\mathcal{A}_{\mathcal{T}}$.

- If $(D \cup d(s, \mathcal{T} \setminus \text{var}(D)))_{\overline{\mathcal{T}}} = \mathsf{sp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(s)$, then $\bigcup_{s' \in p(s,D)} s' = D$ and thus $p(s, \bigcup_{s' \in p(s,D)} s') = p(s,D)$.
- If there exists $\mathrm{v}^{val} \in \mathcal{A}_{\overline{\mathcal{T}}}$ so that $\text{var}(D \cup d(s, \mathcal{T} \setminus \text{var}(D)) \setminus \mathsf{sp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(s)) \cap \overline{\mathcal{T}} = \{\mathrm{v}\}$, then for all state $s' \in p(s,D)$, s' differs from s on the regular variable v and on variables in $\mathcal{T} \setminus \overline{\mathcal{T}}$. Thus, $\bigcup_{s' \in p(s,D)} s' = (D \cup d(s, \mathcal{T} \setminus \text{var}(D))) \setminus \{\mathrm{v}^{val'} \mid \mathrm{v}^{val'} \in s\}$. By construction of p, it comes: $p(s, \bigcup_{s' \in p(s,D)} s') = p(s,D)$ because $\mathrm{v}^{val'} \in s'$ would contradict the condition $|s' \setminus \mathsf{sp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(s)| - |\mathcal{T} \setminus \overline{\mathcal{T}}| = 1$.
- Otherwise, $|\text{var}(D \cup d(s, \mathcal{T} \setminus \text{var}(D)) \setminus \mathsf{sp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(s)) \cap \overline{\mathcal{T}}| > 1$ then there exists two states $s'_1, s'_2 \in p(s,D)$, so that they differ from s on a different regular variable each. Especially, by construction of p, $\mathsf{sp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(s) \subseteq s'_1 \cup s'_2 \subseteq D \cup d(s, \mathcal{T} \setminus \text{var}(D))$. Therefore, $\bigcup_{s' \in p(s,D)} s' \subseteq D \cup d(s, \mathcal{T} \setminus \text{var}(D))$. Finally, and by definition of p, $D \cup d(s, \mathcal{T} \setminus \text{var}(D)) \subseteq \bigcup_{s' \in p(s,D)} s'$ because for each atom in $D \cup d(s, \mathcal{T} \setminus \text{var}(D))$, it is possible to build a state s' containing it:

either as the projection of the initial state s or as the only variable changing its value in s' compared to $\mathsf{sp}_{\overline{\mathcal{F}}\to\overline{\mathcal{T}}}(s)$. In conclusion: $D \cup d(s, \mathcal{T} \setminus \mathrm{var}(D)) = \bigcup_{s' \in p(s,D)} s'$, which gives: $p(s, \bigcup_{s' \in p(s,D)} s') = p(s, D)$.

Thus, $\mathcal{T}_{asyn}(P) = \mathcal{T}_{asyn}(P_{\mathcal{O}}(\mathcal{T}_{asyn}(P)))$, according to Theorem 1. □

C Appendix: Proofs of Section 4

Theorem 3: Properties of Least Revision *Let R be a $\mathcal{M}$VL rule and $s \in \mathcal{S}^{\mathcal{F}}$ such that $R \sqcap s$. Let $S_R := \{s' \in \mathcal{S}^{\mathcal{F}} \mid R \sqcap s'\}$ and $S_{\mathrm{spe}} := \{s' \in \mathcal{S}^{\mathcal{F}} \mid \exists R' \in L_{\mathrm{spe}}(R, s, \mathcal{A}, \mathcal{F}), R' \sqcap s'\}$.*

Let P be a $\mathcal{DM}$VLP and $T, T' \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ such that $|\mathrm{first}(T)| = 1 \wedge \mathrm{first}(T) \cap \mathrm{first}(T') = \emptyset$. The following results hold:

1. $S_{\mathrm{spe}} = S_R \setminus \{s\}$,
2. $L_{\mathrm{rev}}(P, T, \mathcal{A}, \mathcal{F})$ *is consistent with* T,
3. $\overset{P}{\hookrightarrow} T' \implies \overset{L_{\mathrm{rev}}(P,T,\mathcal{A},\mathcal{F})}{\hookleftarrow} T'$,
4. $\overset{P}{\hookrightarrow} T \implies \overset{L_{\mathrm{rev}}(P,T,\mathcal{A},\mathcal{F})}{\hookleftarrow} T$,
5. P *is complete* $\implies L_{\mathrm{rev}}(P, T, \mathcal{A}, \mathcal{F})$ *is complete.*

Proof.

1. First, let us suppose that $\exists s'' \notin S_R \setminus \{s\}$ such that $\exists R' \in L_{\mathrm{spe}}(R, s, \mathcal{A}, \mathcal{F}), R' \sqcap s''$. By definition of matching $R' \sqcap s'' \implies \mathrm{body}(R') \subseteq s''$. By definition of least specialization, $\mathrm{body}(R') = \mathrm{body}(R) \cup \{\mathrm{v}^{val}\}, \mathrm{v}^{val'} \in s, \mathrm{v}^{val} \notin \mathrm{body}(R), val \neq val'$. Let us suppose that $s'' = s$, then $\mathrm{body}(R') \not\subseteq s''$ since $\mathrm{v}^{val} \in \mathrm{body}(R')$ and $\mathrm{v}^{val} \notin s$, this is a contradiction. Let us suppose that $s'' \neq s$ then $\neg(R \sqcap s'')$, thus $\mathrm{body}(R) \not\subseteq s''$ and $\mathrm{body}(R') \not\subseteq s''$, this is a contradiction.
   Second, let us assume that $\exists s'' \in S_R \setminus \{s\}$ such that $\forall R' \in L_{\mathrm{spe}}(R, s, \mathcal{A}, \mathcal{F}), \neg(R' \sqcap s'')$. By definition of S_R, $R \sqcap s''$. By definition of matching $\neg(R' \sqcap s'') \implies \mathrm{body}(R') \not\subseteq s''$. By definition of least specialization, $\mathrm{body}(R') = \mathrm{body}(R) \cup \{\mathrm{v}^{val}\}, \mathrm{v}^{val'} \in s, val \neq val'$. By definition of matching $R \sqcap s'' \implies \mathrm{body}(R) \subseteq s'' \implies s'' = \mathrm{body}(R) \cup I, \mathrm{body}(R) \cap I = \emptyset$ and thus $\mathrm{body}(R') \not\subseteq s'' \implies \mathrm{v}^{val} \notin I$. The assumption implies that $\forall \mathrm{v}^{val'} \in I, \forall R' \in L_{\mathrm{spe}}(R, s, \mathcal{A}, \mathcal{F}), \mathrm{v}^{val} \in \mathrm{body}(R'), val \neq val'$. By definition of least specialization, it implies that $\mathrm{v}^{val'} \in s$ and thus $I = s \setminus \mathrm{body}(R)$ making $s'' = s$, which is a contradiction. Conclusion: $S_{\mathrm{spe}} = S_R \setminus \{s\}$
2. By definition of a consistent program, if two sets of $\mathcal{M}$VL rules SR_1, SR_2 are consistent with T then $SR_1 \cup SR_2$ is consistent with T. Let $R_P = \{R \in P \mid R \sqcap s, \forall (s, s') \in T, \mathrm{head}(R) \notin s'\}$ be the set of rules of P that conflict with T. By definition of least revision $L_{\mathrm{rev}}(P, T, \mathcal{A}, \mathcal{F}) = (P \setminus R_P) \cup \bigcup_{R \in R_P} L_{\mathrm{spe}}(R, s, \mathcal{A}, \mathcal{F})$. The first part of the expression $P \setminus R_P$ is consistent with T since $\nexists R' \in P \setminus R_P$ such that R' conflicts with T. The second part of the expression $\bigcup_{R \in R_P} L_{\mathrm{spe}}(R, s, \mathcal{A}, \mathcal{F})$ is also consistent with T: $\nexists R' \in L_{\mathrm{spe}}(R, s, \mathcal{A}, \mathcal{F}), R' \sqcap s$ thus $\nexists R' \in L_{\mathrm{spe}}(R, s, \mathcal{A}, \mathcal{F})$ that conflict with T and $\bigcup_{R \in R_P} L_{\mathrm{spe}}(R, s, \mathcal{A}, \mathcal{F})$ is consistent with T. Conclusion: $L_{\mathrm{rev}}(P, T, \mathcal{A}, \mathcal{F})$ is consistent with T.
3. Let $(s_1, s_2) \in T'$ thus $s_1 \neq s$. From definition of realization, $\mathrm{v}^{val} \in s_2 \implies \exists R \in P, \mathrm{head}(R) = \mathrm{v}^{val}, R \sqcap s_1$. If $\neg R \sqcap s$ then $R \in L_{\mathrm{rev}}(P, T, \mathcal{A}, \mathcal{F})$ and $\overset{L_{\mathrm{rev}}(P,T,\mathcal{A},\mathcal{F})}{\hookleftarrow} (s_1, s_2)$. If $R \sqcap s$, from the first point $\exists R' \in L_{\mathrm{spe}}(R, s, \mathcal{A}, \mathcal{F}), R' \sqcap s_1$ and since $\mathrm{head}(R') = \mathrm{head}(R) = \mathrm{v}^{val}, \overset{L_{\mathrm{rev}}(P,T,\mathcal{A},\mathcal{F})}{\hookleftarrow} (s_1, s_2)$. Applying this reasoning on all elements of T' implies that $\overset{P}{\hookrightarrow} T' \implies \overset{L_{\mathrm{rev}}(P,T,\mathcal{A},\mathcal{F})}{\hookleftarrow} T'$.

4. Let $(s_1, s_2) \in T$, since $\overset{P}{\hookrightarrow} T$ by definition of realization $\forall \mathrm{v}^{val} \in s_2, \exists R \in P, R \sqcap s_1, \mathrm{head}(R) = \mathrm{v}^{val}$. By definition of conflict, R is not in conflict with T thus $R \in L_{\mathrm{rev}}(P, T, \mathcal{A}, \mathcal{F})$ and $\overset{L_{\mathrm{rev}}(P,T,\mathcal{A},\mathcal{F})}{\hookrightarrow} T$.
5. Let $(s_1, s_2) \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, if P is complete, then by definition of a complete program $\forall \mathrm{v} \in \mathcal{V}, \exists R \in P, R \sqcap s_1, \mathrm{var}(\mathrm{head}(R)) = v$. If $\neg(R \sqcap s)$ then $R \in L_{\mathrm{rev}}(P, T, \mathcal{A}, \mathcal{F})$. If $R \sqcap s$, from the first point $\exists R' \in L_{\mathrm{spe}}(R, s, \mathcal{A}, \mathcal{F}), R' \sqcap s_1$ and thus $R' \in L_{\mathrm{rev}}(P, T, \mathcal{A}, \mathcal{F})$ and since $\mathrm{var}(\mathrm{head}(R')) = \mathrm{var}(\mathrm{head}(R)) = \mathrm{v}$, $L_{\mathrm{rev}}(P, T, \mathcal{A}, \mathcal{F})$ is complete.

□

Proposition 2: Optimal Program of Empty Set $P_{\mathcal{O}}(\emptyset) = \{\mathrm{v}^{val} \leftarrow \emptyset \mid \mathrm{v}^{val} \in \mathcal{A}_{\mathcal{T}}\}$.
Proof. Let $P = \{\mathrm{v}^{val} \leftarrow \emptyset \mid \mathrm{v}^{val} \in \mathcal{A}_{\mathcal{T}}\}$. The $\mathcal{M}$VLP P is consistent and complete by construction. Like all $\mathcal{M}$VLPs, $\overset{P}{\hookrightarrow} \emptyset$ and there is no transition in $\emptyset$ to match with the rules in P. In addition, by construction, the rules of P dominate all $\mathcal{M}$VL rules. □

Proposition 3: From Suitable to Optimal *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. If P is a $\mathcal{DM}$VLP suitable for T, then $P_{\mathcal{O}}(T) = \{R \in P \mid \forall R' \in P, R' \geq R \implies R \geq R'\}$.*
Proof. Since any possible $\mathcal{M}$VL rule consistent with T is dominated, all the rules of the optimal program are dominated. Since the only rules dominating a rule of the optimal program is the rule itself, the optimal program is a subset of any suitable program. If we remove the dominated rules, only remains the optimal program. □

Theorem 4: Least Revision and Suitability *Let $s \in \mathcal{S}^{\mathcal{F}}$ and $T, T' \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ such that $|\mathrm{first}(T')| = 1 \wedge \mathrm{first}(T) \cap \mathrm{first}(T') = \emptyset$. $L_{\mathrm{rev}}(P_{\mathcal{O}}(T), T', \mathcal{A}, \mathcal{F})$ is a $\mathcal{DM}$VLP suitable for $T \cup T'$.*
Proof. Let $P = L_{\mathrm{rev}}(P_{\mathcal{O}}(T), T')$. Since $P_{\mathcal{O}}(T)$ is consistent with T, by Theorem 3, P is also consistent with T and thus consistent with $T' \cup T$. Since $P_{\mathcal{O}}(T)$ realizes T by Theorem 3, $\overset{P}{\hookrightarrow} T$. Since $s \notin \mathrm{first}(T)$, a $\mathcal{M}$VL rule R such that $\mathrm{body}(R) = s$ does not conflict with T. By definition of suitable program $\exists R' \in P_{\mathcal{O}}(T), R' \geq R$, thus $\overset{P_{\mathcal{O}}(T)}{\hookrightarrow} T'$. Since $\overset{P_{\mathcal{O}}(T)}{\hookrightarrow} T'$ by Theorem 3 $\overset{P}{\hookrightarrow} T'$ and thus $\overset{P}{\hookrightarrow} T \cup T'$. Since $P_{\mathcal{O}}(T)$ is complete, by Theorem 3, P is also complete. To prove that P verifies the last point of the definition of a suitable $\mathcal{M}$VLP, let R be a $\mathcal{M}$VL rule not conflicting with $T \cup T'$. Since R is also not conflicting with T, there exists $R' \in P_{\mathcal{O}}(T)$ such that $R' \geq R$. If R' is not conflicting with T', then R' will not be revised and $R' \in P$, thus R is dominated by a rule of P. Otherwise, R' is in conflict with T', thus $R' \sqcap s$ and $\forall (s, s') \in T', \mathrm{head}(R') \notin s'$. Since R is not in conflict with T' and $\mathrm{head}(R) = \mathrm{head}(R')$, since $R' \geq R$ then $\mathrm{body}(R) = \mathrm{body}(R') \cup I, \exists \mathrm{v}^{val} \in I, \mathrm{v}^{val} \notin s$. By definition of least revision and least specialization, there is a rule $R'' \in L_{\mathrm{spe}}(R', s)$ such that $\mathrm{v}^{val} \in \mathrm{body}(R'')$ and since $R'' = \mathrm{head}(R') \leftarrow \mathrm{body}(R') \cup \mathrm{v}^{val}$ thus $R'' \geq R$. Thus R is dominated by a rule of P. □

Theorem 5: GULA Termination, Soundness, Completeness, Optimality *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$.*
(1) *Any call to* **GULA** *on finite sets terminates,*
(2) $\mathbf{GULA}(\mathcal{A}, T, \mathcal{F}, \mathcal{T}) = P_{\mathcal{O}}(T)$,
(3) $\forall \mathcal{A}' \subseteq \mathcal{A}|_{\mathcal{T}}, \mathbf{GULA}(\mathcal{A}_{\mathcal{F}} \cup \mathcal{A}', T, \mathcal{F}, \mathcal{T}) = \{R \in P_{\mathcal{O}}(T) \mid \mathrm{head}(R) \in \mathcal{A}'\}$.
Proof. In this proof we refer to the detailed pseudo-code of **GULA** given in Appendix in Algorithm 5 and Algorithm 6.

(1) The algorithm of **GULA** iterates on finite sets, and thus terminates.

(3) Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. The algorithm iterates over each atom $\mathrm{v}^{val} \in \mathcal{A}'$, $\mathcal{A}' \subseteq \mathcal{A}_{\mathcal{T}}$ iteratively to extract all states s such that $(s, s') \in T \implies \mathrm{v}^{val} \notin s'$. This is equivalent to group the transitions by initial state: generate the set $TT = \{T'_s \subseteq T \mid s \in \mathcal{S}^{\mathcal{F}}, \mathrm{first}(T'_s) = \{s\} \wedge \forall s' \in \mathcal{S}^{\mathcal{T}}, (s, s') \in T \implies (s, s') \in T'_s\}$.

To prove that $\forall \mathcal{A}' \subseteq \mathcal{A}_{\mathcal{T}}, \mathbf{GULA}(\mathcal{A}_{\mathcal{F}} \cup \mathcal{A}', T, \mathcal{F}, \mathcal{T}) = \{R \in P_{\mathcal{O}}(T) \mid \mathrm{head}(R) \in \mathcal{A}'\}$ and thus $\mathbf{GULA}(\mathcal{A}, T, \mathcal{F}, \mathcal{T}) = P_{\mathcal{O}}(T)$, it suffices to prove that the main loop (Algorithm 5, lines 23–50) preserves the invariant $P^{val}_{\mathrm{v}} = \{R \in P_{\mathcal{O}}(T_i) \mid \mathrm{head}(R) = \mathrm{v}^{val} \in \mathcal{A}'\}$ after the

i^{th} iteration where T_i is the union of all set of transitions of TT already selected line 23 after the i^{th} iteration for all i from 0 to $|TT|$.

Line 22 initializes $P_{\mathrm{v}^{val}}$ to $\{\mathrm{v}^{val} \leftarrow \emptyset\}$. Thus by Proposition 2, after line 22, $P_{\mathrm{v}^{val}} = \{R \in P_{\mathcal{O}}(\emptyset) \mid \text{head}(R) = \mathrm{v}^{val}\}$.

Let us assume that before the $(i+1)^{\text{th}}$ iteration of the main loop, $P_{\mathrm{v}^{val}} = \{R \in P_{\mathcal{O}}(T_i) \mid \text{head}(R) = \mathrm{v}^{val}\}$. Through the loop of lines 25–28, $P' = \{R \in P_{\mathcal{O}}(T_i) \mid R \text{ does not conflict with } T_{i+1} \wedge \text{head}(R) = \mathrm{v}^{val}\}$ is computed. Then the set $P'' = \bigcup_{R \in P_{\mathcal{O}}(T_i) \setminus P' \wedge \text{head}(R) = \mathrm{v}^{val}} L_{\text{spe}}(R, s, \mathcal{A}, \mathcal{F})$ is iteratively build through the calls to **least_specialization** (Algorithm 6) at line 31 and the dominated rules are pruned as they are detected by the loop of lines 32–49. Each revised rule can be dominated by a rule in $\{R \in P_{\mathcal{O}}(T_i) \setminus P'\}$ or another revised rule and thus dominance must be checked from both. But only a revised rule ($R \in P''$) can be dominated by a revised rule: if a rule in $\{R \in P_{\mathcal{O}}(T_i) \setminus P'\}$ is dominated by a revised rule, then it was dominated by its original rule in $\{R \in P_{\mathcal{O}}(T_i)\}$ which is impossible since $P_{\mathrm{v}^{val}} = \{R \in P_{\mathcal{O}}(T_i) \mid \text{head}(R) = \mathrm{v}^{val}\}$. Thus it is safe to only check domination of the revised rules by previous rules ($P_{\mathcal{O}}(T_i) \setminus P'$) or by other revised rules (P''). Thus by Theorem 4 and Proposition 3, $P_{\mathrm{v}^{val}} = \{R \in P_{\mathcal{O}}(T_{i+1}) \mid \text{head}(R) = \mathrm{v}^{val}\}$ after the $(i+1)^{\text{th}}$ iteration of the main loop. By induction, at the end of all the loop lines 23–50, $P_{\mathrm{v}^{val}} = \{R \in P_{\mathcal{O}}(\bigcup_{T' \in TT} T') \mid \text{head}(R) = \mathrm{v}^{val}\} = \{R \in P_{\mathcal{O}}(T) \mid \text{head}(R) = \mathrm{v}^{val}\}$ since it has iterated on all elements of TT. Since the same operation holds for each $\mathrm{v}^{val} \in \mathcal{A}'$, $P = \bigcup_{\mathrm{v}^{val} \in \mathcal{A}'} P_{\mathrm{v}^{val}} = \{R \in P_{\mathcal{O}}(T) \mid \text{head}(R) = \mathrm{v}^{val} \wedge \mathrm{v}^{val} \in \mathcal{A}'\}$ after all iterations of the loop of line line 6. Finally: $\forall \mathcal{A}' \subseteq \mathcal{A}_{\mathcal{T}}, \mathbf{GULA}(\mathcal{A}_{\mathcal{F}} \cup \mathcal{A}', T, \mathcal{F}, \mathcal{T}) = \{R \in P_{\mathcal{O}}(T) \mid \text{head}(R) \in \mathcal{A}'\}$.

(2) Thus $\mathbf{GULA}(\mathcal{A}, T, \mathcal{F}, \mathcal{T}) = \mathbf{GULA}(\mathcal{A}_{\mathcal{F}} \cup \mathcal{A}_{\mathcal{T}}, T, \mathcal{F}, \mathcal{T}) = \{R \in P_{\mathcal{O}}(T) \mid \text{head}(R) \in \mathcal{A}_{\mathcal{T}}\} = P_{\mathcal{O}}(T)$. □

Theorem 6: GULA Complexity *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ be a set of transitions, Let $n := max(|\mathcal{F}|, |\mathcal{T}|)$ and $d := \max(\{|\mathsf{dom}(\mathrm{v})|) \in \mathbb{N} \mid \mathrm{v} \in \mathcal{F} \cup \mathcal{T}\}$. The worst-case time complexity of* **GULA** *when learning from T belongs to $\mathcal{O}(|T|^2 + |T| \times (2n^4 d^{2n+2} + 2n^3 d^{n+1}))$ and its worst-case memory use belongs to $\mathcal{O}(d^{2n} + 2nd^{n+1} + nd^{n+2})$.*

Proof. Let $d_f := \max(\{|\mathsf{dom}(\mathrm{v})| \in \mathbb{N} \mid \mathrm{v} \in \mathcal{F}\})$ (resp. $d_t := \max(\{|\mathsf{dom}(\mathrm{v})| \in \mathbb{N} \mid \mathrm{v} \in \mathcal{T}\})$) be the maximal number of values of features (resp. target) variables. The algorithm takes as input a set of transition $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ bounding the memory use to $O(d_f^{|\mathcal{F}|}) \times d_t^{|\mathcal{T}|}) = O(d^{2n})$. The learning is performed iteratively for each possible rule head $\mathrm{v}^{val} \in \mathcal{A}' \subseteq \mathcal{A}_{\mathcal{T}}$. The extraction of negative example requires to compare each transition of T one to one and thus has a complexity of $op_1 = O(|T|^2)$. Those transitions are stored in $Neg_{\mathrm{v}^{val}}$ which size is at most $|\mathcal{S}^{\mathcal{F}}|$ extending the memory use to $O(d_f^{|\mathcal{F}|} \times d_t^{|\mathcal{T}|} + d_f^{|\mathcal{F}|})$ which is bounded by $O(d^{2n} + d^n)$.

The learning phase revises a set of rule $P_{\mathrm{v}^{val}}$ where each rule has the same head v^{val}. There are at most $d_f^{|\mathcal{F}|} \leq d^n$ possible rule bodies and thus $|P_{\mathrm{v}^{val}}| \leq d_t^{|\mathcal{F}|} \leq d^n$, the memory use of $|P_{\mathrm{v}^{val}}|$ is then $O(d_t^{|\mathcal{F}|})$ extending the memory bound to $O(d_f^{|\mathcal{F}|} \times d_t^{|\mathcal{T}|} + d_f^{|\mathcal{F}|}) + d_f^{|\mathcal{F}|}) = O(d_f^{|\mathcal{F}|} \times d_t^{|\mathcal{T}|} + 2d_f^{|\mathcal{F}|}))$, which is bound by $O(d^{2n} + 2d^n)$.

For each state s of $Neg_{\mathrm{v}^{val}}$, each rule of $P_{\mathrm{v}^{val}}$ that matches s are extracted into a set of rules R_m. This operation has a complexity of $op_2 = O(d_f^{|\mathcal{F}|} \times |\mathcal{F}|)$ bound by $O(nd^n)$. Each rule of R_m are then revised using least specialization, this operation has a complexity of $O(|\mathcal{F}|^2)$ bound by $O(n^2)$. $|R_m| \leq d_f^{|\mathcal{F}|} \leq d^n$ thus the revision of all matching rules is $op_3 = O(d_f^{|\mathcal{F}|} \times n^2)$ bounded by $O(d^n \times n^2)$. All revisions are stored in LS and there are at most $d_f \times |\mathcal{F}| \leq dn$ revisions for each rule, thus $|LS| \leq d_f^{|\mathcal{F}|} \times d_f |\mathcal{F}| \leq d^n \times dn$ extending the memory bound to $O(d_f^{|\mathcal{F}|} \times d_t^{|\mathcal{T}|} + 2d_f^{|\mathcal{F}|}) + d_f |\mathcal{F}| \times d_f^{|\mathcal{F}|})$ bounded by $O(d^{2n} + 2d^n + nd^{n+1})$.

Learning is performed for each $\mathrm{v}^{val} \in \mathcal{A}' \subseteq \mathcal{T}$, thus the memory usage of **GULA** is therefore $O(d_f^{|\mathcal{F}|} \times d_t^{|\mathcal{T}|} + |\mathcal{A}'|(2d_f^{|\mathcal{F}|} + d_f |\mathcal{F}| \times d_f^{|\mathcal{F}|}))$, bounded by $O(d_f^{|\mathcal{F}|} \times d_t^{|\mathcal{T}|} + td_t(2d_f^{|\mathcal{F}|}) + d_f |\mathcal{F}| \times d_f^{|\mathcal{F}|}))$ wich is bounded by $O(d^{2n} + dn(2d^n + nd^{n+1})) = O(d^{2n} + 2nd^{n+1} + nd^{n+2})$.

The worst-case memory use of **GULA** is thus $O(d^{2n} + 2nd^{n+1} + nd^{n+2})$.

All rules of LS are compared to the rule of $P_{\mathrm{v}^{val}}$ for domination check, this operation has a complexity of $op_4 = O(2 \times |LS| \times |P_{\mathrm{v}^{val}}| \times |\mathcal{F}|^2) = O(2 \times d_f^{|\mathcal{F}|} \times d_f |\mathcal{F}| \times d^n \times n^2) = O(2 \times |\mathcal{F}|^3 \times d_f^{2|\mathcal{F}|+1})$ which is bounded by $O(2 \times n^3 \times d^{2n+1})$.

Learning is performed for each $\mathrm{v}^{val} \in \mathcal{A}' \subseteq \mathcal{T}$, $|\mathcal{A}'| \leq |\mathcal{T}| d_t$, thus the complexity is bound by $O(op_1 + |T| \times |\mathcal{T}| \times d_t(op_2 + op_3 + op_4)) = O(|T|^2 + |T|\ times|\mathcal{T}| \times d_t(d_f^{|\mathcal{F}|} \times |\mathcal{F}| + d_f^{|\mathcal{F}|} \times n^2 + 2 \times |\mathcal{F}|^3 \times d_f^{2|\mathcal{F}|+1}))$ which is bounded by $O(|T|^2 + |T| \times nd(d^n \times n^2 + d^n \times n^2 + 2 \times n^3 \times d^{2n+1})) = O(|T|^2 + |T| \times nd(2n^3 d^{2n+1} + 2n^2 d^n)) = O(|T|^2 + |T| \times (2n^4 d^{2n+2} + 2n^3 d^{n+1}))$.

The computational complexity of **GULA** is thus $O(|T|^2 + |T| \times (2n^4 d^{2n+2} + 2n^3 d^{n+1}))$. □

D Appendix: Proofs of Section 5

Theorem 7: Optimal $\mathcal{DM}$VLP and Constraints Correctness Under Synchronous Constrained Semantics *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, it holds that $T = \mathcal{T}_{syn-c}(P_{\mathcal{O}}(T) \cup C'_{\mathcal{O}}(T))$.*
Proof. From Definition 9, $\forall (s, s') \in T, s' \subseteq \mathsf{Conclusions}(s, P_{\mathcal{O}}(T))$ thus according to Definition 21, $s' \in \mathcal{T}_{syn-c}(P_{\mathcal{O}}(T))(s)$, thus $T \subseteq \mathcal{T}_{syn-c}(P_{\mathcal{O}}(T))$ (property 1).

By Definition 24, $\forall (s, s') \in T, \nexists C \in C_{\mathcal{O}}(T), C \sqcap (s, s')$, thus since $C'_{\mathcal{O}}(T) \subseteq C_{\mathcal{O}}(T), \nexists C \in C'_{\mathcal{O}}(T), C \sqcap (s, s')$ and then $T \subseteq \mathcal{T}_{syn-c}(P_{\mathcal{O}}(T) \cup C'_{\mathcal{O}}(T))$ (property 2).

Let us suppose $\exists (s, s') \in \mathcal{T}_{syn-c}(P_{\mathcal{O}}(T) \cup C'_{\mathcal{O}}(T)), (s, s') \notin T$. From Definition 21, $\forall \mathrm{v}^{val} \in s', \exists R \in P_{\mathcal{O}}(T), \mathrm{body}(R) \sqcap s, \mathrm{head}(R) = \mathrm{v}^{val}$. From Definition 24, $\exists C' \in C_{\mathcal{O}}(T), C' \sqcap (s, s')$ since $(s, s') \notin T$. But since $\exists (s, s') \in \mathcal{T}_{syn-c}(P_{\mathcal{O}}(T) \cup C'_{\mathcal{O}}(T))$, thus $C' \notin C'_{\mathcal{O}}(T)$. From Definition 25, it implies that $\exists \mathrm{v}^{val} \in s', \nexists R \in P_{\mathcal{O}}(T), \mathrm{head}(R) = \mathrm{v}^{val}, \forall \mathrm{w} \in \mathcal{F}, \forall val', val'' \in \mathsf{dom}(\mathrm{w}), \mathrm{w}^{val'} \in \mathrm{body}(R) \wedge \mathrm{w}^{val''} \in \mathrm{body}(C) \implies val' = val''$. Since $\mathrm{body}(C') \subseteq (s \cup s')$, $\nexists R \in P_{\mathcal{O}}(T), \mathrm{head}(R) = \mathrm{v}^{val}, \mathrm{body}(R) \subseteq s$, thus $s' \not\subseteq \mathsf{Conclusions}(s, P_{\mathcal{O}}(T))$ and by Definition 21, $(s, s') \notin \mathcal{T}_{syn-c}(P_{\mathcal{O}}(T) \cup C'_{\mathcal{O}}(T))$, contradiction, thus $\mathcal{T}_{syn-c}(P_{\mathcal{O}}(T) \cup C'_{\mathcal{O}}(T)) \subseteq T$ (property 3).

From property 2 and 3: $\mathcal{T}_{syn-c}(P_{\mathcal{O}}(T) \cup C'_{\mathcal{O}}(T)) = T$. □

Theorem 8: Synchronizer Correctness *Given any set of transitions T,* **Synchronizer***$(\mathcal{A}, T, \mathcal{F}, \mathcal{T})$ outputs $P_{\mathcal{O}}(T) \cup C'_{\mathcal{O}}(T)$.*
Proof. Let $G1 = GULA(\mathcal{A}, T, \mathcal{F}, \mathcal{T})$ and $G2 = GULA(\mathcal{A}_{\mathcal{F} \cup \mathcal{T} \cup \{\varepsilon^1\}}, T', \mathcal{F} \cup \mathcal{T}, \{\varepsilon\})$. From Theorem 5, $P = G1 = P_{\mathcal{O}}(T)$ (property 1).

Let $P' = G2$. By definition of T': $\forall (s, s') \in T', s' = \{\varepsilon^0\}$. Thus $\forall R \in P'$, R is consistent with T' by Theorem 5, thus $\nexists (s, s') \in T', R \sqcap s$, since $\mathrm{head}(R) = \varepsilon^1$ because $\forall (s, s') \in T', s' = \{\varepsilon^0\}$ (property 2).

From Theorem 5, $P' = \{R \in P_{\mathcal{O}}(T') \mid \mathrm{head}(R) = \varepsilon^1\}$. From Definition 9, $P_{\mathcal{O}}(T')$ is complete thus $\forall (s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}, ss' := s \cup s', ss' \notin \mathrm{first}(T'), \exists R \in P', R \sqcap ss'$ (property 3).

From definition of T', $(s, s') \in T \implies (s \cup s', \{\varepsilon^0\}) \in T'$, thus $\forall C \in P', C$ is a constraint (property 4).

- From property 2 and 4: $(s, s') \in T \implies (s \cup s', \{\varepsilon^0\}) \in T' \implies \nexists C \in P', C \sqcap (s, s')$, P' consistent with T.
- From property 3 and 4: $(s, s') \notin T \implies (s \cup s') \notin \mathrm{first}(T') \implies \exists R \in P', R \sqcap (s, s')$, P' is complete with T.
- If there exists a constraint consistent with T that is not dominated by a constraint in P' it implies that a rule consistent with T' whose head is ε^1 is not dominated by a rule in $G2$ wich is in contradiction with Theorem 5. All constraint consistent with T are dominated by a constraint in P'.
- From Theorem 5, the rules of $G2$ do not dominate eachover, thus the same hold for the constraint of P'.
- From Definition 24, $P' = C_{\mathcal{O}}(T)$ (property 5).

Now let us prove that $P'' = C'_{\mathcal{O}}(T)$. Let us suppose that $P'' \neq C'_{\mathcal{O}}(T)$. Since $P'' \subseteq C_{\mathcal{O}}(T)$, according to Definition 25, therefore P'' is missing a useful optimal constraint ($C'_{\mathcal{O}}(T) \setminus P'' \neq \emptyset$), or contains a useless optimal constraint ($P'' \setminus C'_{\mathcal{O}}(T) \neq \emptyset$).

1) Suppose that $C \notin P''$ but $C \in C'_{\mathcal{O}}(T)$, meaning that P'' misses a useful constraint C. Since $C \in C'_{\mathcal{O}}(T)$, $\exists (s, s') s \xrightarrow{P_{\mathcal{O}}(T)} s'$, $C \sqcap (s, s')$. Since $s \xrightarrow{P_{\mathcal{O}}(T)} s'$, according to Definition 5 $\exists S \subseteq P_{\mathcal{O}}(T), s' = \{\text{head}(R) \mid R \in S\} \wedge \forall R \in S, R \sqcap s$. By Definition 20, $C \subseteq s \cup s'$ thus $\text{body}(C) \cap \mathcal{A}_{\mathcal{F}} \subseteq s$ and $\text{body}(C) \cap \mathcal{A}_{\mathcal{T}} \subseteq s'$. By definition of C_{rules}, $\forall \text{v}^{val} \in \text{body}(C) \cap \mathcal{A}_{\mathcal{T}}, \forall R \in S, \big(\text{var}(\text{head}(R)) = \text{v} \wedge \text{head}(R) \in \text{body}(C) \implies R \in C_{rules}(\text{v})\big)$ and since $s \xrightarrow{P_{\mathcal{O}}(T)} s'$, $\forall \text{v} \in C_{targets}, C_{rules}(\text{v}) \neq \emptyset$. Thus there exists a *combi* such that $\forall \text{v} \in \mathcal{F}, |\{\text{v}^{val} \in \text{body}(R) \mid val \in \mathsf{dom}(\text{v}) \wedge R \in combi\}| \leq 1$, contradiction.

2) Suppose that $C \notin C'_{\mathcal{O}}(T)$ but $C \in P''$, meaning that P'' contains a useless constraint C. Thus, $\{(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}} \mid s \xrightarrow{P_{\mathcal{O}}(T)} s' \wedge C \sqcap (s, s')\} = \emptyset$. Since $C \in P''$ there is a *combi* such that $|\{\text{v}^{val} \in \text{body}(R) \mid val \in \mathsf{dom}(\text{v}) \wedge R \in combi\}| \leq 1$, thus $\exists s \in \mathcal{S}^{\mathcal{F}}, \text{body}(C) \cap \mathcal{A}_{\mathcal{T}} \subseteq s \wedge \forall R \in combi, R \sqcap s$. Let $S := \{s' \in \mathcal{S}^{\mathcal{T}} \mid s \xrightarrow{P_{\mathcal{O}}(T)} s'\}$. Because $P_{\mathcal{O}}(T)$ is complete, $S \neq \emptyset$. Since $\forall R \in combi, R \in P_{\mathcal{O}}(T), \exists s' \in S, \forall R \in combi, \text{head}(R) \in s'$. Since $\text{body}(C) \cap \mathcal{A}_{\mathcal{T}} = \{\text{head}(R) \mid R \in combi\} \subseteq s'$, $C \sqcap (s, s')$.

Thus $P'' = C'_{\mathcal{O}}(T)$ (property 6).

From property 1 and 6, $Synchronizer(\mathcal{A}, T, \mathcal{F}, \mathcal{T}) = P_{\mathcal{O}}(T) \cup C'_{\mathcal{O}}(T)$.

□

Theorem 8: Synchronizer Complexity *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ be a set of transitions, let $n := max(|\mathcal{F}|, |\mathcal{T}|)$ and $d := \max(\{|\mathsf{dom}(\text{v})| \in \mathbb{N} \mid \text{v} \in \mathcal{F} \cup \mathcal{T}\})$ and $m := |\mathcal{F}| + |\mathcal{T}|$.*

The worst-case time complexity of **Synchronizer** *when learning from T belongs to $\mathcal{O}((d^{2n} + 2nd^{n+1} + nd^{n+2}) + (|T|^2 + |T| \times (2m^4 d^{2m+2} + 2m^3 d^{m+1})) + (d^{n^n}))$ and its worst-case memory use belongs to $\mathcal{O}((d^{2n} + 2nd^{n+1} + nd^{n+2}) + (d^{2m} + 2md^{m+1} + md^{m+2}) + (nd^n))$.*

Proof. Let $d_f := \max(\{|\mathsf{dom}(\text{v})| \in \mathbb{N} \mid \text{v} \in \mathcal{F}\})$ (resp. $d_t := \max(\{|\mathsf{dom}(\text{v})| \in \mathbb{N} \mid \text{v} \in \mathcal{T}\})$) be the maximal number of values of features (resp. target) variables. Let $n := max(|\mathcal{F}|, |\mathcal{T}|)$ and $d := \max(\{|\mathsf{dom}(\text{v})| \in \mathbb{N} \mid \text{v} \in \mathcal{F} \cup \mathcal{T}\})$ and $m := |\mathcal{F}| + |\mathcal{T}|$. The first call to **GULA** has complexity of $\mathcal{O}(|T|^2 + |T| \times (2n^4 d^{2n+2} + 2n^3 d^{n+1}))$ and the memory is bound by $\mathcal{O}(d^{2n} + 2nd^{n+1} + nd^{n+2})$ according to Theorem 6.

Computing $T' := \{(s \cup s', \{\varepsilon^0\}) \mid (s, s') \in T\}$ has a linear complexity of $\mathcal{O}(|T|)$. The call $GULA(\mathcal{A}_{\mathcal{F} \cup \mathcal{T} \cup \{\varepsilon^1\}}, T', \mathcal{F} \cup \mathcal{T}, \{\varepsilon\})$ considers target variables as features variables to learn constraints, i.e., the body of constraints can have m conditions. Thus the complexity of this call to **GULA** is bound by $\mathcal{O}(|T'|^2 + |T'| \times (2m^4 d^{2m+2} + 2m^3 d^{m+1})) = \mathcal{O}(|T|^2 + |T| \times (2m^4 d^{2m+2} + 2m^3 d^{m+1}))$ since $|T'| = |T|$ and the memory is bound by $\mathcal{O}(d^{2m} + 2md^{m+1} + md^{m+2})$ according to Theorem 6.

To discard useless constraints, Algorithm 3 searches for a set of rules that can be applied at the same time as the constraint: first it extract the constraint target variables $C_{targets} := \{\text{v} \in \mathcal{T} \mid \exists val \in \mathsf{dom}(\text{v}), \text{v}^{val} \in \text{body}(C)\}$ and search for compatible rules with the constraint $\forall \text{v} \in C_{targets}, C_{rules}(\text{v}) := \{R \in P \mid \text{var}(\text{head}(R)) = \text{v} \wedge \text{head}(R) \in \text{body}(C) \wedge \forall \text{w} \in \mathcal{F}, \forall val, val' \in \mathsf{dom}(\text{w}), \big(\text{w}^{val} \in \text{body}(R) \wedge \text{w}^{val'} \in \text{body}(C)\big) \implies val = val'\}$. The constraint contains at most $|\mathcal{T}|$ target conditions. For each target variable, there is at most $d_f^{|F|}$ rules in P. Thus, computing the Cartesian product of rules grouped by head variables has a time complexity of $\mathcal{O}(d_f^{|F|^{|\mathcal{T}|}})$ which is bound by $\mathcal{O}(d^{n^n})$ and a memory complexity of $\mathcal{O}(|P|)$ which is bound by $\mathcal{O}(nd^n)$.

The computational complexity of **Synchronizer** is thus $\mathcal{O}((d^{2n} + 2nd^{n+1} + nd^{n+2}) + (|T|^2 + |T| \times (2m^4 d^{2m+2} + 2m^3 d^{m+1})) + (d^{n^n}))$ and its memory is bound by $\mathcal{O}((d^{2n} + 2nd^{n+1} + nd^{n+2}) + (d^{2m} + 2md^{m+1} + md^{m+2}) + (nd^n))$. □

E Appendix: Proofs of Section 6

Proposition 5: Uniqueness of Impossibility-Optimal Program *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. The $\mathcal{DM}$VLP impossibility-optimal for T is unique and denoted $P_{\mathcal{O}}(T)$.*
Proof. Same proof than for Proposition 1 by replacing "suitable" by "impossibility-suitable". □

F Appendix: detailed pseudo-code of Section 4

Algorithms 5 and 6 provide the detailed pseudocode of **GULA**. Algorithm 5 learns from a set of transitions T the conditions under which each value *val* of each variable v may appear in the next state. Here, learning is performed iteratively for each value of variable to keep the pseudo-code simple. But the process can easily be parallelized by running each loop in an independent thread, bounding the run time to the variable for which the learning is the longest. In the case where we are not interested about the dynamics of some variables, the parameter $\mathcal{A}'$ and $\mathcal{T}'$ can be reduced accordingly. The algorithm starts by the pre-processing of the input transitions. Lines 7–18 of Algorithm 5 correspond to the extraction of $Neg_{\mathrm{v}^{val}}$, the set of all negative examples of the appearance of v^{val} in next state: all states such that v never takes the value *val* in the next state of a transition of T. For efficiency purpose, it is important that the negatives examples are ordered in a way that reduce the difference between nearby elements, for example lexicographically. Indeed, it increase the proportion of revised rules (produced to satisfy a previous example) still consistent with the following examples, reducing the average number of rules stored and thus checked in the following processes. Those negative examples are then used during the following learning phase (lines 21–50) to iteratively learn the set of rules $P_{\mathcal{O}}(T)$. The learning phase starts by initializing a set of rules $P_{\mathrm{v}^{val}}$ to $\{R \in P_{\mathcal{O}}(\emptyset) \mid \mathrm{head}(R) = \mathrm{v}^{val}\} = \{\mathrm{v}^{val} \leftarrow \emptyset\}$ (see Proposition 2).

$P_{\mathrm{v}^{val}}$ is iteratively revised against each negative example *neg* in $Neg_{\mathrm{v}^{val}}$. All rules R_m of $P_{\mathrm{v}^{val}}$ that match *neg* have to be revised. In order for $P_{\mathrm{v}^{val}}$ to remain optimal, the revision of each R_m must not match *neg* but still matches every other state that R_m matches. To ensure that, the least specialization (see Definition 17) is used to revise each conflicting rule R_m. Algorithm 6 shows the pseudo code of this operation. For each variable of $\mathcal{F}'$ so that $\mathrm{body}(R_m)$ has no condition over it, a condition over another value than the one observed in state *neg* can be added (lines 3–8). None of those revision match *neg* and all states matched by R_m are still matched by at least one of its revisions.

Each revised rule can be dominated by a rule in $P_{\mathrm{v}^{val}}$ or another revised rules and thus dominance must be checked from both. But only revised rule can be dominated by a revised rule: if a rule in $P_{\mathrm{v}^{val}}$ is dominated by a revised rule, then it was dominated by its original rule and thus could not be part of $P_{\mathrm{v}^{val}}$ since it would have been discard in a previous step. Thus we can safely only check the revised rules to discard the ones dominated by the new current revised rule. The non-dominated revised rules are then added to $P_{\mathrm{v}^{val}}$.

Once $P_{\mathrm{v}^{val}}$ has been revised against all negatives example of $Neg_{\mathrm{v}^{val}}$, $P_{\mathrm{v}^{val}} = \{R \in P_{\mathcal{O}}(T) \mid \mathrm{head}(R) = \mathrm{v}^{val}\}$, that is, $P_{\mathrm{v}^{val}}$ is the subset of rules of the final optimal program having v^{val} as head. Finally, $P_{\mathrm{v}^{val}}$ is added to P and the loop restarts with another atom. Once all values of each variable have been treated, the algorithm outputs P which is then equal to $P_{\mathcal{O}}(T)$.

Algorithm 5 **GULA**($\mathcal{A}'$,T,$\mathcal{F}'$,$\mathcal{T}'$, *learning_mode*)

INPUT: A set of atoms $\mathcal{A}'$, a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}'} \times \mathcal{S}^{\mathcal{T}'}$, two sets of variables $\mathcal{F}'$ and $\mathcal{T}'$, a string $learning\_mode \in \{$"$possibility$", "$impossibility$"$\}$.
OUTPUT: $P_{\mathcal{O}}(T)$ if $learning\_mode =$ "$possibility$" or $\overline{P_{\mathcal{O}}(T)}$ if $learning\_mode =$ "$impossibility$".

$T := \{(s_1, \{s_2 \mid (s_1, s_2) \in T\}) \mid s_1 \in \text{first}(T)\}$ // Group transitions by initial state
$T := sort(T)$ // Sort the transitions in Lexicographical order over feature states
$P := \emptyset$
for each $\text{v}^{val} \in \mathcal{A}'$ such that $\text{v} \in \mathcal{T}'$ **do**
  // 1) Extraction of positives and negative examples of possibility
  $Pos_{\text{v}val} := \emptyset$
  $Neg_{\text{v}val} := \emptyset$
  for each $(s_1, S) \in T$ **do**
    $negative\_example := true$
    for each $s_2 \in S$ **do**
      if $\text{v}^{val} \in s_2$ **then**
        $negative\_example := false$
        $Pos_{\text{v}val} := Pos_{\text{v}val} \cup \{s_1\}$
        break
    if $negative\_example == true$ **then**
      $Neg_{\text{v}val} := Neg_{\text{v}val} \cup \{s_1\}$
  if $learning\_mode ==$ "$impossibility$" **then**
    $Neg_{\text{v}val} = Pos_{\text{v}val}$ // Positive examples of possibility are negatives examples of impossibility.
  // 2) Revision of the rules of v^{val} to avoid matching of negative examples
  $P_{\text{v}val} := \{\text{v}^{val} \leftarrow \emptyset\}$
  for each $neg \in Neg_{\text{v}val}$ **do**
    $M := \emptyset$ // Set of rules of $P_{\text{v}val}$ that are in conflict
    for each $R \in P_{\text{v}val}$ **do** // Extract all rules that conflict and remove them from P
      if body(R) $\subseteq neg$ **then**
        $M := M \cup \{R\}$
        $P_{\text{v}}^{val} := P_{\text{v}}^{val} \setminus \{R\}$
    $LS := \emptyset$
    for each $R_m \in M$ **do** // Revise each conflicting rule
      $P' := least\_specialization(R_m, neg, \mathcal{A}', \mathcal{F}')$
      for each $R_{ls} \in P'$ **do**
        $dominated := false$
        for each $R_p \in P_{\text{v}val}$ **do** // Check if the revision is dominated by $P_{\text{v}val}$
          if body(R_p) $\subseteq$ body(R_{ls}) **then**
            $dominated := true$
            break
        if $dominated == true$ **then**
          continue
        for each $R_p \in LS$ **do** // Check if the revision is dominated by LS
          if body(R_p) $\subseteq$ body(R_{ls}) **then**
            $dominated := true$
            break
        if $dominated == true$ **then**
          continue
        for each $R_p \in LS$ **do**// Remove previous specialization that are now dominated
          if body(R_{ls}) $\subseteq$ body(R_p) **then**
            $LS := LS \setminus \{R_p\}$
        $LS := LS \cup \{R_{ls}\}$ // Add the revision
      $P_{\text{v}val} := P_{\text{v}val} \cup LS$ // Add non-dominated revisions
  $P := P \cup P_{\text{v}val}$
return P

Algorithm 6 **least_specialization**(R, s, $\mathcal{A}'$, $\mathcal{F}'$) : specialize R to avoid matching of s

INPUT: a rule R, a state s, a set of atoms $\mathcal{A}'$ and a set of variables $\mathcal{F}'$
OUTPUT: a set of rules LS which is the least specialization of R by s according to $\mathcal{F}'$ and $\mathcal{A}'$.

$LS := \emptyset$
// Revise the rules by least specialization
for each $\mathrm{v}^{val} \in s$ **do**
  if $\mathrm{v} \notin \mathrm{var}(\mathrm{body}(R))$ **then** // Add condition for all values not appearing in s
    for each $\mathrm{v}^{val'} \in \mathcal{A}', \mathrm{v} \in \mathcal{F}', val' \neq val$ **do**
      $R' := \mathrm{head}(R) \leftarrow (\mathrm{body}(R) \cup \{\mathrm{v}^{val'}\})$
      $LS := LS \cup \{R'\}$
return LS

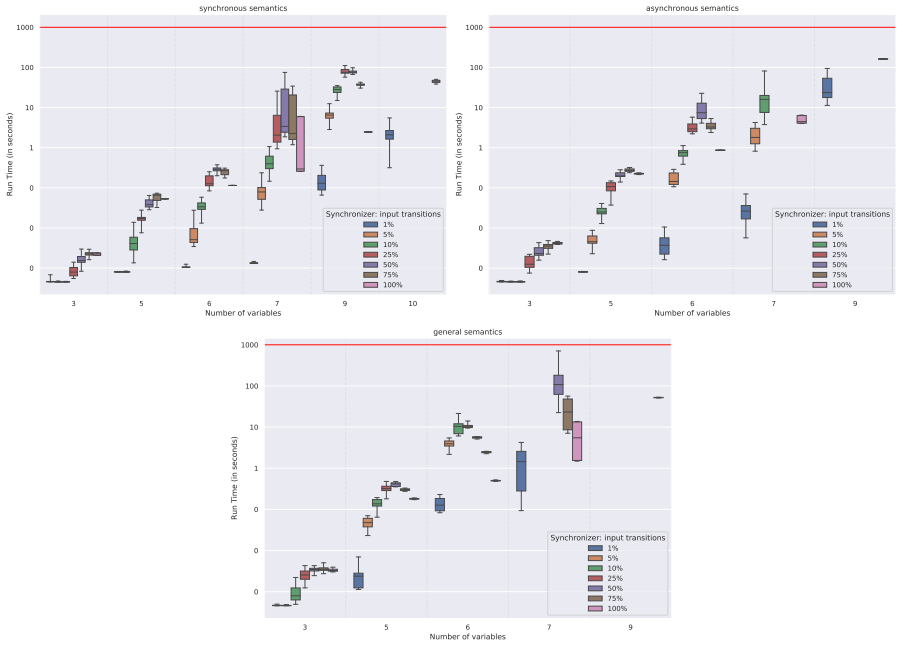


Fig. 13: Run time of **Synchronizer** from a random set of $1\%, 5\%, 10\%, 25\%, 50\%, 75\%, 100\%$ of the transitions of a Boolean network from Boolenet and PyBoolNet with size varying from 3 to 10 variables. Time out is set at $1,000$ seconds and 10 runs where performed for each setting.

G Synchronizer Scalability

Figure 13 shows the run time of **Synchronizer** when learning from transitions of Boolean networks from Boolenet [11] and PyBoolnet [27] with same settings as in the experiements of Table 3. For the synchronous and general semantics, it is only when we are given a subset of all possible transitions that the algorithm output constraints, since all combination of heads of matching rules are allowed for those two semantics. Those constraint at least prevent transitions from unseen states and also some combination of atoms that are missing in next states but that are observed individually. Even when it outputs an empty set of constraint, the learning process needs to produce and revises constraint until its no more possible, so run time of full set of transitions is also considered. In the asynchronous case, given a set of transitions T, it needs to learn the constraints ensuring at most one change per transitions, i.e., $\{\xleftarrow{\perp} a_t^i, b_t^j, a_{t-1}^{i'}, b_{t-1}^{j'} \mid a, b \in \mathcal{A}_{\overline{T}}, i \neq i' \wedge j \neq j'\}$ and the ones preventing the projection when only one variable can be updated: $\{C \mid \{a_t^i, a_{t-1}^i\} \in \text{body}(C), a \in \mathcal{A}_{\overline{T}}, \nexists (s, s') \in T, \text{body}(C) \subseteq s \cup s'\}$. Those second kind of constraint will be specific to the few states this limitation occurs and show the limits of propositional representation for the explanation of the dynamics.

Learning constraints is obviously more costly than learning regular rules since both features and targets variables can appear in the body, i.e., number of features becomes $|\mathcal{F}|+|\mathcal{T}|$. The algorithm reached the time out of $1,000$ seconds with benchmarks of 10 nodes for synchronous semantics and 7 nodes for asynchronous and general semantics. Scalability of the algorithm can be greatly improved by using the approximated version of **GULA** for

learning both rules and constraints. If learning rules can be done in polynomial time, learning constraints remains exponential. Since we do not present this approximated algorithm in this paper we will not go into the details. In short, this approximated version needs positives examples and thus require to generate the Cartesian product of all applicable rules heads for each initial state observed which is exponential. Scalability, readability and applicability could be improved by considering first order generalization of both rule and constraints but those generalization are application dependant and thus remains as future work. Such generalization is required to perform proper prediction from unseen states, thus application of the synchronizer output for prediction from unseen states are out of the scope of this paper.

H Information About this Paper

H.1 History of the paper

This paper is a substantial extension of [43] where a first version of **GULA** was introduced. In [43], there was no distinction between feature and target variables, i.e., variables at time step t and $t+1$. From this consideration, interesting properties arise and allow to characterize the kind of semantics compatible with the learning process of the algorithm (Theorem 1). It also allows to represent constraints and to propose an algorithm (**Synchronizer**, Section 5) to learn programs whose dynamics can mimic any given set of transitions with optimal properties on both rules and constraints. It also allows to use **GULA** to learn human readable explanations in form of rules on static classification problems (as long as all variables are discrete), which will be one of the focus of our future works.

H.2 Main contributions of the paper

The main contributions of this paper are:

- A modeling of discrete memory-less dynamics system as multi-valued propositional logic. This modeling is independent of the dynamical semantics the system relies on, as long as it respects some given properties we provided in this paper. The main contributions of this formalism is the characterization of optimality and the study of which semantics are compatible with this formalism (which includes notably synchronous, asynchronous and general semantics).
- A first algorithm named **GULA**, to learn such optimal programs.
- The formalism is also extended to represent and use constraints. This allows to reproduce any discrete memory-less dynamical semantics behaviors inside the logic program when the original semantics is unknown.
- A second algorithm named **Synchronizer**, that exploits **GULA** to learn a logic program with constraints that can reproduce any given set of state transitions. The method we proposed is able to learn a whole system dynamics, including its semantics, in the form of a single propositional logic program. This logic program not only explains the behavior of the system in the form of human readable propositional logic rules but also is able to reproduce the behavior of the observed system without the need of knowing its semantics. Furthermore, the semantics can be explained, without any previous assumption, in the form of human readable rules inside the logic program. In other words, the approach allows to learn all the previously cited semantics, as well as new ones.
- A heuristic method allowing to use **GULA** to learn a model able to predict from unseen case.
- Evaluation of these methods on benchmarks from biological litterature regarding scalability, prediction accuracy and explanation quality.

H.3 What evidence is provided

We show through theoretical results the correctness of our approach for both modeling and algorithms (see above contribution for details). Empirical evaluation is performed on benchmarks coming from biological literature. It shows the capacity of **GULA** to produce correct models when all transitions are available. Also, we observe that learned models generalize to unseen data when given a partial input in those experiments.

H.4 Related work

The paper refers to relevant related work. As we discussed in the related work section, our approach is quite related to Bain and Srinivasan [3], Evans *et al.* [12,13], Katzouris *et al.* [23], Fages [14].

The techniques we propose in this paper are a continuation of the works on the LFIT framework from [20,45,43].

In [19,21], state transitions systems are represented with logic programs, in which the state of the world is represented by an Herbrand interpretation and the dynamics that rule the environment changes are represented by a logic program P. The rules in P specify the next state of the world as an Herbrand interpretation through the *immediate consequence operator* (also called the T_P *operator*) [54,2] which mostly corresponds to the synchronous semantics we present in Section 3. In this paper, we extend upon this formalism to model multi-valued variables and any memory-less discrete dynamic semantics including synchronous, asynchronous and general semantics.

[20] proposed the LFIT framework to learn logic programs from traces of interpretation transitions. The learning setting of this framework is as follows. We are given a set of pairs of Herbrand interpretations (I, J) as positive examples such that $J = T_P(I)$, and the goal is to induce a *normal logic program* (NLP) P that realizes the given transition relations. As far as we know, this concept of *learning from interpretation transition* (LFIT) has never been considered in the ILP literature before [20]. In this paper, we propose two algorithms that extend upon this previous work: **GULA** to learn the minimal rules of the dynamics from any semantics states transitions that respect Theorem 1 and **Synchronizer** that can capture the dynamics of any memory-less discrete dynamic semantics.

I Declarations

I.1 Funding

This work was supported by JSPS KAKENHI Grant Number JP17H00763 and by the "Pays de la Loire" Region through RFI Atlanstic 2020.

I.2 Conflicts of interest/Competing interests

None

I.3 Availability of data and material

Experiments data and sources code is available at `https://github.com/Tony-sama/pylfit` under GPL-3.0 License.

I.4 Code availability

Algorithms and experiments sources code is available at `https://github.com/Tony-sama/pylfit` under GPL-3.0 License.

J Response to Reviewers

We thank the reviewers for their care in providing feedback and suggestions to improve the paper. We apologize for the increased number of pages in the paper, but the fix and modifications needed by the reviewers' requests required these additions.

Summary of updates:

- Majors
  - Insertion of examples for every definition
  - Section 6 links theory and evaluation parts by providing formally how to use GULA to learn from partial observations
  - Explainability experiements (Section 7.3)
  - Trivials baselines for all experiments
  - Updated synchronizer algorithm and proofs (cleaning of useless constraint)
- Minors
  - Methodological comparison with Progol/Aleph (Section 8)
  - Clarified notations
  - Experiments results tables replaced by boxplots graphs

In the following, we provide comments on how every remark from the panel of reviewers has been taken into account.

J.1 Reviewer 1

Thank you for your comments, we fixed the typos you found and addressed your claims as follows.

Section 3.1 (semantics): I found this section less well written, a bit verbose and with non optimal notations. I don't have specific suggestions to improve it however.

We tried to clarify the readibility of this section by adding examples and figures.

p13l29: point attractors are not necessarily known by the reader.

We added a sentence to define the term informally when it first appears (Section 3.1).

p14l8 At this point I was puzzled by the absence of reference to ILP. Probably such a reference should be done beforehand in the introduction.

We have added some sentences in the introduction section to make the positioning of our approach explicit in the ILP community. In particular, we make the bridge with two survey papers about ILP. The more detailed comparison with several ILP approaches appear in the related works (Section 8).

p21l15 Please comment the extra complexity coming basically from $|T| + |F|$ instead of $max(|T|, |F|)$.

We added details about the extra complexity that arise when learning constraints. Since a constraint body can also contain target atoms, their body size can be up to $|T| + |F|$ where a $\mathcal{DM}$VLP rule body is bound by $|F|$. In the complexity we use $n = max(|T|, |F|)$ to simplify the formula, but as stated in the proof it is the possible size of rule bodies that most influences the complexity, the proof of complexity given in appendix has more details.

Section 7.2 The different approaches should probably be mentioned in different paragraphs (probability distribution, optimization, ILP, constraint solving, ASP, event calculus). Furthermore, although that section is focused on the applicative side (e.g. ref [41] using ILP), a succinct comparison from the algorithmic point of view (in terms of difference of computation or complexity when applied to the instances satisfying the same hypotheses) would be interesting to know to better position the contribution from a methodological point of view.

We dispatched the different approaches in different paragraphs as advised. We added an algorithmic comparison with Progol and Aleph in the Related Works section (Section 8).

J.2 Reviewer 2

Thank you for your comments, we fixed the typos you found and addressed your claims as follows.

In fact, my main concern is the presentation, which suffers from heavy notation. Take as examples Definitions 15 and 25. I do not see how one can understand them in a reasonable amount of time, keeping in mind all the necessary notations and definitions like sp, Ccl, $d_{\bar{T}}$, etc. I have the impression that the concepts are actually quite simple. However, they got blurred in the vast number of definitions and clumsy notation. For that reason, I would like to ask the authors to make a serious attempt to simplify the notation, to reduce the number of definitions, and to underpin both by many more examples. On a minor note, there are many grammar mistakes, in particular singular vs. plural. The paper should be revised carefully in that regard.

We clarified notations and added examples for every definitions.

Detailed comments:

- page 8, line 15: Do you mean for all MVL rules R from P?

It refers to any MVL rules, not only the one in P, we clarified it into "for any possible MVL rule R"

- page 9: I could not understand Theorem 2. Should it be something like "If, for all P, ... then, for all P", rather than "For all P, if ..."? Also, remove $\wedge$ at the end of item (1).

The first and last P are not the same indeed, we corrected it and revised the presentation of the theorem.

- page 9, line 42: I suppose val at t-1 and val at t are not necessarily the same? So wouldn't it be better to distinguish them?

They are indeed the same value, on different variables (v_t and v_{t-1}). We clarified in the text.

- page 10, line 31: Definition 13 would really be easier to grasp by means of an example.

We added an example. Figure 4 also serves as an explanation for this definition.

- page 12, line 21: What do you mean by $d(s, \emptyset) = \emptyset$ is always valid? Shouldn't this occur in Definition 14 as well?

We mean that $d(s, \emptyset) = \emptyset$ is an admissible value for d in all cases. We clarified in the text.

- page 17, Theorem 7: Could you please discuss the complexity wrt. a very naive algorithm that just checks all possible logic programs that "arise" from T (I guess there are only finitely many candidate programs)?

The number of possible programs is all subsets of possible rules which would have factorial complexity to generate, then rule consistency and minimality can be checked in polynomial time regarding input transitions, but completeness requires to check that all possible initial states are matched by a rule for each target variables, having exponential cost. According to the definition of an optimal program, we can find it using a brute force enumeration of all possible rules, check their individual consistency against the input transitions and remove dominated rules. We added this trivial algorithm at the end of the $\mathcal{DM}$VLP section (Algorithm 1) and discuss its complexity against GULA (Section 4.2). We also use it as a new baseline for the scalability experiments (Figure 9). GULA is of higher complexity than the brute force but in practice GULA is much faster as shown by the experiments.

- page 17, line 44: Maybe there is a better notation for a constraint than $\leftarrow b(C)$, as it is very close to a rule.

We added a $\perp$ sign on the arrow for the notation of a constraint to distinguish it from other rules. However, a constraint is still a special case of a MVL rule.

- page 19: Please format Definition 25 in a readable way.

This definition has been fixed and rewritten.

J.3 Reviewer 3

Thank you for your comments, we fixed the typos you found and addressed your claims as follows.

- weak connection between the theoretical results and experimental evaluation

We added a section (Section 6) about learning from partial observation, i.e., realistic data that makes the link between the theoretical parts and the experiments.

- lack of conclusive experimental results.

We added more details about accuracy experiments and added an explainability evaluation. Experiments are conclusive, GULA can learn a predictive models even from few observations as shown in accuracy experiments. The program learned are also meaningful as shown in explainability experiments, i.e., the rules learned are more and more similar to the optimal ones when given more examples and even with a few observations, the program learned is near-optimal.

- some issues with the presentation of the results which makes it difficult to judge the significance of the results (e.g. on accuracy).

We replaced the table form of the results for boxplots, we hope it is more readable. We also compare the results of GULA to several trivial baselines.

As mentioned above the main contributions of the paper appear to be 3 and 4 above as 1 and 2 were published before, though this paper also improves and extends previous results, e.g. new accuracy experiments. These contributions are supported by the theoretical results (propositions, theorems etc) and experimental evaluation. However, I think the connections between the theory and evaluation is not very clear. Also, the experimental results should be improved & clarified so that the paper is acceptable for MLJ publication.

Connection is now made with a new section on learning from partial data and experiments improved and clarified.

For example, the significance of the accuracy results (Section 6.2 & Fig 6) are not very clear, especially as there is no comparison with any baseline (default accuracy) or any other approach. Also, the paper mentions that: "If one is only interested by the prediction, it is certainly easier to achieve better results using other methods like neural networks or random forest since prediction here is basically a binary classification" and that "In the case where explainability is of interest, the rules used for the predictions and their weights may be quite simple human readable candidates for explanations." However, the paper doesn't include any example rule learned for any of the datasets. So, all of these make it difficult to judge the significant of the experimental results.

We added baselines for all experiments and new experiments regarding explainability.

As mentioned above, there are some issues with the presentation & discussion of the experimental results. The theoretical framework seems to be appropriate, however I haven't checked the correctness of all proofs in the paper (and there are more in the appendix). There are some discussions on the previous version of this work and its limitations. However, the limitation of the proposed approach in this paper should be also discussed in more details. The experimental results and the connections with the theory should be also clarified.

We added more discussion about limitations in the evaluation section and related work.

I think the paper is heavy with maths notations and definitions and difficult to follow due to the lack of examples, i.e there are 24 definitions, 19 theorems and 6 propositions, but only 4 examples in the paper.

We added examples for all definitions.

The Tables & Figures representing he results could be improved. Table 1 & 2: the numbers in the table are not clear and very difficult to compare, better to use a graph. The run time graphs used in [35] seem to be much clear.

We replaced tables results by boxplots.

Section 6.2 & Fig 6: The significance of the accuracy results are not very clear, especially as there is no comparison with the baseline (default accuracy) or any other approach.

We added several trivial baselines.

Also the single run approach could be extended to average curves. I.e. "10% to 90% are chosen randomly to form a training set and the rest for a test .." I think it is better to repeat the random selections, e.g 10 runs, and plotting the average curve with error bars instead of single run with a randomly selected test and training set.

This is what was actually done, but it was not explained well enough: each of the 10 runs generate a different training and test set at random. We added an example with a figure to clarify the experimental settings, and the table showing only the average of all runs has

been replaced with boxplot graphs that show the distribution of the individual runs grouped by settings.

In general, I think not a well organised experimental section, e.g some details of experimental methods described after the results.

We reorganised the Experiments section and moved some of its content to other sections that were more relevant.

I think the current title is a bit misleading: Learning "any semantic" for dynamic systems sounds like a strong claim, while in fact the paper is proposing a formalism that could potentially be used for modelling synchronous, asynchronous and "general semantics" as defined in the paper. The results don't support the "any semantic" claim, or at least "any semantic" should be defined (is this the same as "general semantics" ?)

We clarified the title. What we consider "any semantics" is actually the set of discrete memoryless semantics, that are all captured by the Synchronizer algorithm (see Theorem 7). Any pseudo-idempotent dynamical semantics is still handled by GULA (see Theorem 1). The general semantics (see Definition 16) is just a semantics among others pseudo-idempotent semantics, which can be seen as more general than the classical synchronous and asynchronous ones.

Please include more examples to clarify the definitions & theorems in the theoretical framework (there are 24 definitions, 19 theorems and 6 propositions, but only 4 examples in the paper). Please also include examples of learned rules in the evaluation section (sec 6.2) to exemplify the claim that "In the case where explainability is of interest, the rules used for the predictions and their weights may be quite simple human readable candidates for explanations.".

We added examples for all definitions. The new section about $\mathcal{WDM}$VLP contains examples of explanations predictions, evaluated in new explainability experiments in evaluation section.

Section 6.2 & Fig 6: Please clarify the significance of the accuracy results, i.e. there is no comparison with the baseline (default accuracy) or any other approach.

We added more details and baselines comparisons.

Table 1 & 2: the numbers in the table are not clear and very difficult to compare. Better to use a graph, please consider adopting a similar approach to the timing graphs in [35].

We replaced them by boxplots.

Please re-organise the experimental section, e.g details of experimental methods should be described before the results.

We re-organised the experimental sections accordingly and moved some content to more relevant sections.

The significance results and the limitation of the proposed approach in this paper should be discussed in more details.

We added more discussion on this matter in evaluation part, related work and conclusions.

Please consider alternative title or clarify "any semantic" in the title. Is this the same as "general semantics" ?

We clarified the title into "any memory-less discrete semantics". This is not the same as general semantics, please see above for details.

Please clarify (at the beginning) the main contributions of this paper wrt the conference paper [35] and the (practical) significance of the results. Also, please try to clarify the connections between the theoretical and the experimental results.

We highlighted in the introduction the major improvements w.r.t. the previous conference paper where **GULA** was introduced for the first time. We also added some information about the significance from a practical point of view. We clarified the connection between theoretical and experimental results by adding a new section on learning from partial data and new experiments regarding explanations.

Sec 6.2: "10% to 90% are chosen randomly to form a training set and the rest for a test .." It might be better to repeat the random selections, e.g 10 runs, and plotting the average curve with error bars instead of single run with a randomly selected test and training set

we actually already do so but it needed clarifying, please see above.

Also, in Fig 6, the numbers on the X axis 0.1, 0.2, 0.3, ... appear to be proportion rather than "Percentage" as in the label (?)

We corrected the figures legend accordingly.

J.4 Reviewer 4

Thank you for your comments, we fixed the typos you found and addressed your claims as follows.

This paper presents 2 algorithms for learning propositional logic programs from state transitions. The logic program models the dynamics of a system, for example a biological system. There are various semantics and the authors focus on three: the synchronous, asynchronous and general semantics. In general, the semantics of a logic program (as presented in this paper) is a set of transitions between the current state (atoms that have specific values) and a potential next state. In this way, logic programs can capture potential nondeterminism of the system. The objective of the two algorithms presented in this paper is to induce a logic program that explain a set of observations (state transitions) under one of the three semantics.

To clarify, the first algorithm GULA can handle the three semantics and many others as stated in Theorem 1. The second algorithm, the Synchronizer, can handle any discrete memory-less dynamics semantics as stated in Theorem 7. So both algorithm do not only cover the three aforementioned semantics.

Even though I find that the paper presents an interesting approach, I think that the paper lacks the clarity to be published as-is in this journal.

We hope to have improved the clarity of the paper by making important changes to it: new examples, better organization and new and more detailed experiments.

The paper is, in general, readable but in some section heavy on notation. Moreover, in my opinion it lacks more examples to communicate some crucial concepts (see the detailed comments). The complexity of notation increased substantially in comparison with the similar definitions of [35]. As a result, it did not give me confidence that the theoretical results can easily follow. Some of the proofs seems to be out of sync with the main text of the paper. Overall, I think the paper needs more polishing and several changes to increase readability (see my detailed comments below).

We clarified some notations into more readable ones like body(R) in place of just b(R), etc. Moreover we added examples for all definitions.

The authors evaluate their approach in terms of scalability and quality of predictability. They consider some examples from boolean networks and they achieve to learn a logic program with 12 variables. However, the time complexity result states that is exponential on d (the size of the domain) which, in those experiments is trivially $d = |dom(0, 1)| = 2$.

We added brute force experiments to compare with our algorithm GULA (see Section 7) and we can see it's not trivial to obtain the optimal rules even for 12 Boolean variables. Although our method can tackle multivalued models (with more than 2 discrete values for variables), most biological models are Boolean (2 discrete values). Moreover, Boolean models are already enough to observe the combinatorial explosion of possible rules even on small models, as the complexity is at least $(2 + 1)^n$ for a model with n variables: each variable can appear as 0, 1 or not appear in the body of a rule.

page 11, line 30: Figure 4 is not mentioned in the main text and I find it difficult to interpret. The authors make an effort to give an intuition of the semantics but I think this was done in a very abstract level. It would be helpful, to have an example based on Figure 4: what is the logic program of the boolean network and the state transitions given each semantics. This is also crucial in order to understand Figure 5.

We added the corresponding programs, the transitions are given in the transitions diagrams, we clarified the way to read the diagrams.

page 14, line 27-29: It is not clear to me why the produced set of rules match all the states of R except s. It is obvious why it will not match s since the atom $v^{val} \notin s$ is added to the body of the rule but it is not obvious why $\{s' \mid s' \in S^F \wedge R \sqcap s' \wedge v^{val} \notin s'\} = \{s\}$. Later, on page 16, line 24 that property of L_{spe} is assumed.

This is formally shown in proof of the least revision Theorem 4 in appendix Section C. Intuitively, for each variable not in s, we take a v^{val} so that $v^{val} \notin s$ and add it to R to form

a new rule. Doing so for each variable and values creates a set of rules, each with exactly one more atom in the body compared to R. Because $s' \neq s$, there is (at least) an atom in s' that is not in s nor in R. Thus, this atom has been added by one of the above steps to form a new rule, which matches s'.

page 16, line 7: The definition of $Neg_{v_{val}}$ is strange. Is s range over $S^{F'}$? I think the correct definition should be $Neg_{v_{val}} = \{s \mid (s, s') \in T \wedge v^{val} \notin s'\}$.

Indeed, s ranges over $S^{F'}$ since $(s, s') \in T \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. However, our definition of $Neg_{v_{val}}$ is correct since there can be multiple transitions from the same feature state s (because of non-determinism): an atom does need to appear in all target states to be considered possible, thus a negative example is a feature state from which we never observe v^{val} in next state (not only in one state).

page 18, definition 21: I was expecting a default function 'd' as in Definition 14. That raises the question: If there are no constraints in P, will $T_{sync}(P) = T_{sync-c}(P)$ because the definitions are quite different.

This semantics is meant to exactly follow the optimal constrained logic program; since this program is complete it does not need a default to produce complete target states. Furthermore, a default could change the behavior given by the program which is not desirable here. Thus, it is equivalent to the synchronous semantics with $\forall D \subseteq \mathcal{V}, default(s, D) = \emptyset$ when using a complete program like the optimal program for example.

page 34, theorem 6: The proof is more of a sketch than a formal mathematical proof. Moreover, it does not follow the definitions presented in Algorithm 1 and even mentions things that are not defined like "least_specialization" probably remnants from previous version of the paper. It refers to the detailed pseudo code of GULA given in the appendix, we added reference to it, sorry for the confusion. These are not remnants of previous version of the paper.

page 6, 2 last lines: I think the explanation of "regular variables" confuses than clarify things especially the phrase that appear in both T and F which seems at first sight contradictory with Definition 2. Moreover, if I am not missing it, I think the concept of "regular variables" has not been introduced before that phrase.

We clarified the explanation.

page 10, line 27: I cannot understand the "which is equivalent to" part.

Both are equivalent and mean that the regular variables have the same value. Because of the bijection, and the fact that these functions ignore the stimuli and observation variables, both formulations can be used.