Machine Learning manuscript No.
(will be inserted by the editor)

Learning from interpretation transition

Katsumi Inoue · Tony Ribeiro · Chiaki Sakama

Received: date / Accepted: date

Abstract We propose a novel framework for learning normal logic programs from transitions of interpretations. Given a set of pairs of interpretations (I, J) such that $J = T_P(I)$, where T_P is the immediate consequence operator, we infer the program P. The learning framework can be repeatedly applied for identifying Boolean networks from basins of attraction. Two algorithms have been implemented for this learning task, and are compared using examples from the biological literature. We also show how to incorporate background knowledge and inductive biases, then apply the framework to learning transition rules of cellular automata.

Keywords dynamical systems · Boolean networks · cellular automata · attractors · supported models · learning from interpretation · Inductive Logic Programming

1 Introduction

There is a growing interest in learning dynamics of systems in the field of inductive logic programming (ILP) [32] with applications in planning, scheduling, robotics, bioinformatics, and adaptive and complex systems. In the view that a logic program is a state transition system [20,23], given an Herbrand interpretation representing a current state of the world,

This research was supported in part by the NII research project on "Dynamic Constraint Networks" and by the "Systems Resilience" project at Research Organization of Information and Systems, Japan

K. Inoue
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
E-mail: inoue@nii.ac.jp

T. Ribeiro
Department of Informatics, The Graduate University for Advanced Studies (Sokendai)
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
E-mail: tony_ribeiro@nii.ac.jp

C. Sakama
Department of Computer and Communication Sciences, Wakayama University
Sakaedani, Wakayama 640-8510, Japan
E-mail: sakama@sys.wakayama-u.ac.jp

a logic program P specifies how to define the next state of the world as an Herbrand interpretation through the *immediate consequence operator* (also called the T_P *operator*) [46, 6]. Based on this idea, we here propose a framework to learn logic programs from traces of interpretation transitions.

The learning setting is as follows. We are given a set of pairs of Herbrand interpretations (I, J) such that $J = T_P(I)$ as positive examples, and the goal is to induce a *normal logic program* (NLP) P that realizes the given transition relations. As far as the authors know, this concept of *learning from interpretation transition* (LFIT) has never been considered in the ILP literature. In fact, LFIT is different from any method to learn Boolean functions that has been developed in the field of computational learning theory [25] in the sense that LFIT learns dynamics of systems, while the conventional learning setting is not involved in dynamics. A closer setting can be found in *learning from interpretations* (LFI) [14], in which positive examples are given as Herbrand models of a target program, but again the goal of LFI is not to learn dynamics of systems. Learning action theories [31, 33, 21, 45, 12, 38] can also be related with LFIT, but its goal is not exactly the same as that of LFIT. In particular, LFIT can learn dynamics of systems with *positive and negative feedbacks*, which have not been much taken into account in the literature. Relational reinforcement learning [16] can consider feedbacks in the learning process as rewards, but LFIT learns how such feedbacks can be represented logically by state transition rules. Learning NLPs rather than definite programs has been considered in ILP, e.g., [39], but most approaches do not take the LFI setting. Moreover, from the semantical viewpoint, our framework can learn NLPs under the *supported model semantics* [6] rather than the stable model semantics [19].

An intended direct application of LFIT is learning transition or update rules in *dynamical systems* such as *Boolean networks* [24] and *cellular automata* [47], which have been respectively used as mathematical models of genetic networks and complex adaptive systems. It has been observed that the T_P operator for an NLP P precisely captures the synchronous update of the corresponding Boolean network, where each gene and its regulation function correspond to a ground atom and the set of ground rules with the atom in their heads, respectively [20]. Then, given an input Herbrand interpretation I, which corresponds to a *gene activity profile* (GAP) with gene disruptions for false atoms in I and gene overexpressions for true atoms in I, the interactions between genes are experimentally analyzed by observing an output GAP J such that $J = T_P(I)$ is assumed to hold after a time step has passed. In this setting, LFIT of an NLP P corresponds to inferring a set of gene regulation rules that are complete for those experiments of 1-step GAP transitions. Such a learning task has been analyzed in the literature [3, 4], but no ILP technique has been applied to the problem. Besides, 2-state cellular automata, in which each cell can take either 1 or 0 as a possible value, are instances of Boolean networks, so that their state transitions are determined by the T_P operator [8]. Hence it should be possible to apply LFIT for their learning tasks. Learning transition rules (called *identification*) of cellular automata has been studied in the literature [1, 2], but again no previous work has employed ILP techniques on this problem.

It is known that any trajectory from a GAP in a Boolean network reaches an *attractor*, which is either a fixed point or a periodic oscillation. Then, we can consider a realistic situation to use LFIT, in which the input is a set of trajectories reaching to attractors and the output is a Boolean network, i.e., an NLP, realizing them. In this paper, we will thus show two supposed usages of LFIT: **LF1T** takes 1-step transitions, and **LFBA** assumes trajectories to attractors. Moreover, two algorithms for **LF1T** have been implemented, and are compared using examples of gene regulatory networks in the biological literature. We also suggest how to incorporate background knowledge and inductive biases in LFIT, then apply the whole framework to learning transition rules of cellular automata.

The rest of this paper is organized as follows. Section 2 reviews the logical background of this work, and Section 3 shows how the semantics of logic programs is related to state transitions of dynamical systems. Section 4 introduces **LF1T** together with two versions of its algorithms and proves their correctness. Section 5 considers **LFBA** as variations of **LF1T** and incorporates background knowledge and inductive biases. Section 6 shows experimental results of two versions of **LF1T** on learning Boolean networks and cellular automata. Section 7 discusses related work, and Section 8 concludes the paper.

2 Normal Logic Programs

We consider a first-order language and denote the Herbrand base (the set of all ground atoms) as $\mathcal{B}$. A (*normal*) *logic program* (NLP) is a set of *rules* of the form

$$A \leftarrow A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n \tag{1}$$

where A and A_i's are atoms ($n \geq m \geq 0$). For any rule R of the form (1), the atom A is called the *head* of R and is denoted as $h(R)$, and the conjunction to the right of $\leftarrow$ is called the *body* of R. We represent the set of literals in the body of R of the form (1) as $b(R) = \{A_1, \ldots, A_m, \neg A_{m+1}, \ldots, \neg A_n\}$, and the atoms appearing in the body of R positively and negatively as $b^+(R) = \{A_1, \ldots, A_m\}$ and $b^-(R) = \{A_{m+1}, \ldots, A_n\}$, respectively. An NLP P is called a *definite program* if $b^-(R) = \emptyset$ for every rule R in P. The set of ground instances of all rules in a logic program P is denoted as $ground(P)$. An NLP P is called an *acyclic program* [5] if, for every rule of the form (1) in $ground(P)$, $|A| > |A_i|$ holds for every $i = 1, \ldots, n$ and for some function $|\ \ | : \mathcal{B} \to \mathbb{N}$ (called a *level mapping*) from the Herbrand base to natural numbers.

An (*Herbrand*) *interpretation* I is a subset of $\mathcal{B}$, and is called an (*Herbrand*) *model* of P if I *satisfies* all ground rules from P, that is, for any rule $R \in ground(P)$, $b^+(R) \subseteq I$ and $b^-(R) \cap I = \emptyset$ imply $h(R) \in I$.

An Herbrand interpretation $I \in 2^{\mathcal{B}}$ is *supported* in an NLP P if for any ground atom $A \in I$, there exists a rule $R \in ground(P)$ such that $h(R) = A$, $b^+(R) \subseteq I$, and $b^-(R) \cap I = \emptyset$. I is a *supported model* of P if I is a model of P and is supported in P [6]. It is known that the supported models of P are precisely the models of $Comp(P)$, which is the Clark's *completion* of P [11]. Every acyclic program has the unique supported model [5], but there may be no, one or multiple supported models of an NLP in general.

Given an NLP P and an Herbrand interpretation I, the *reduct* of P relative to I is defined as the definite program: $P^I = \{(h(R) \leftarrow \bigwedge_{B \in b^+(R)} B) \mid R \in ground(P), b^-(R) \cap I = \emptyset\}$. An Herbrand model I is a *stable model* [19] of P if I is the least model of P^I. Since $P^I = P$ holds for any definite program P and any Herbrand interpretation I, the unique stable model of a definite program is its least model.

Both the stable model semantics and the supported model semantics have been major semantics in the field of logic programming. It is known that every stable model is a supported model [29], but not vice versa. For example, the NLP $\{p \leftarrow p,\ q \leftarrow \neg p\}$ has the supported models $\{p\}$ and $\{q\}$, but only the latter is its stable model. Every acyclic program has the unique stable model that is the same as its supported model [5].

For a logic program P and an Herbrand interpretation I, the *immediate consequence operator* (or T_P *operator*) [6] is the mapping $T_P : 2^{\mathcal{B}} \to 2^{\mathcal{B}}$:

$$T_P(I) = \{\, h(R) \mid R \in ground(P),\, b^+(R) \subseteq I,\, b^-(R) \cap I = \emptyset \}. \tag{2}$$

If P is definite, T_P is monotone, i.e., $I_1 \subseteq I_2$ implies $T_P(I_1) \subseteq T_P(I_2)$ [46]. When P is an NLP, however, T_P is generally nonmonotone [6]. Then, I is a model of P iff $T_P(I) \subseteq I$. By definition, I is supported iff $I \subseteq T_P(I)$. Hence, I is a supported model of P iff $T_P(I) = I$. Thus, the T_P operator is more directly connected to the supported model semantics than to the stable model semantics. Note that T_P is *deterministic*, that is, it determines a unique interpretation $T_P(I)$ for any interpretation I. A sequence of applications of the operator on Herbrand interpretations is called an *orbit* [8]. Given a logic program P and an Herbrand interpretation I, the *orbit* of I with respect to the T_P operator is the sequence $\langle {T_P}^k(I)\rangle_{k\in\omega}$, where ${T_P}^0(I) = I$ and ${T_P}^{k+1}(I) = T_P({T_P}^k(I))$ for $k \in \omega$ (ω is a limit ordinal).

3 Representing Dynamics in Logic Programs

Here we consider logic-based representation of dynamical systems, which is a key issue for inductive learning of them. In ILP, a first-order representation is used for a relational concept, and we simply follow this line of research, e.g., [32]. In particular, we do not propose any new learning scheme for generalization and abstraction which are not directly related to dynamics. For instance, if a particle A and a particle B have the same physical properties, then a rule to decide the position of A after a perturbation is added must be the same as a rule for B with the same kind of perturbation. Then, identification of such a rule involves the dynamics, but the names A and B are not crucial so that we can generalize them to be a variable in a common rule. We thus assume that any ILP method can be applied to generalize such individuals, and will focus on learning of dynamics itself in this paper. We here show two such representations to deal with dynamics: One is based on a first-order notation with the time argument, and the other does not use the time argument.

Symbolic representation of dynamic changes has been studied in knowledge representation in AI such as situation calculus [30] and event calculus [26], which are mostly suitable for virtual action sequences. In real-world applications, however, the state of the world changes concurrently from time to time, and all elements in the world may change often *synchronously*. Then, to represent discrete time directly in the simplest way, we can use the time argument in a relational representation: For each relation $p(\mathbf{x})$ among the objects, where p is a predicate and $\mathbf{x}$ is a tuple of its arguments, we can consider its state at time t as $p(\mathbf{x}, t)$. In this way, we shall represent any atom $A = p(\mathbf{x})$ at time t by putting the time argument of the predicate as $A^t = p(\mathbf{x}, t)$. Then, a rule in an NLP of the form (1) can be made a dynamic rule in the first-order expression of the form:

$$A^{t+1} \leftarrow {A_1}^t \wedge \cdots \wedge {A_m}^t \wedge \neg {A_{m+1}}^t \wedge \cdots \wedge \neg {A_n}^t. \tag{3}$$

The rule (3) means that, if $A_1, \ldots, A_m$ are all true at time step t and $A_{m+1}, \ldots, A_n$ are all false at the same time step t, then A is true at the next time step $t + 1$. Note that this kind of dynamic rules is first-order even if the original rule is propositional. Then, any first-order NLP that is a set of rules of the form (3) becomes an acyclic program, in which the stable model semantics and the supported model semantics coincide. Moreover, we can simulate state transition of Boolean networks using this representation and the T_P operator [20].

A *Boolean network* [24] is a pair $N = (V, F)$, where $V = \{v_1, \ldots, v_n\}$ is a finite set of nodes (n is the number of nodes) and $F = \{f_1, \ldots, f_n\}$ is a corresponding set of Boolean functions. The value of node v_i at time step t is denoted as $v_i(t)$. The value of v_i at the next time step $t + 1$ is then determined by $v_i(t + 1) = f_i(v_{i_1}(t), \ldots, v_{i_k}(t))$, where $v_{i_1}, \ldots, v_{i_k}$ are the input nodes to v_i. A *state* of N at time step t is $(v_1(t), \ldots, v_n(t))$, and represents a gene activity profile (GAP) at t when applied to a gene regulatory network. A *trajectory*

of N is a sequence of states obtained by a series of state transitions. As $|V|$ is finite, every trajectory always reaches to some *attractor* [24, 18, 20], which is either a fixed point (called a *point attractor*) or a periodic oscillation (called a *cycle attractor*). A state that reaches an attractor $\mathcal{S}$ is said to belong to the *basin of attraction* of $\mathcal{S}$. Inoue [20] shows a translation of a Boolean network N into an NLP $\tau(N)$ such that $\tau(N)$ is a set of rules of the form (3): For each $v_i \in V$, convert its Boolean function $f_i(v_{i_1}(t), \ldots, v_{i_k}(t))$ into a DNF formula[1] $\bigvee_{j=1}^{l_i} {B_{i,j}}^t$, where $B_{i,j}$ is a conjunction of literals, then generate l_i rules with ${v_i}^{t+1}$ as the head and ${B_{i,j}}^t$ as a body for each $j = 1, \ldots, l_i$. Given a state $S(t) = (v_1(t), \ldots, v_n(t))$ at time step t, let $J^t = \{{v_i}^t \mid v_i \in V,\ v_i(t) \text{ is true in } S(t)\}$. Then the translation τ has the property that the trajectory of N from an initial state $S(0) = (v_1(0), \ldots, v_n(0))$ can be precisely simulated by the sequence of interpretations, $J^0, J^1, \ldots, J^k, J^{k+1}, \ldots$, where $J^{k+1} = T_{\tau(N)}(J^k) \cap \{{v_i}^{t+1} \mid v_i \in V\}$ for $k \geq 0$ [20].

Example 1 Consider the Boolean network $N_1 = (V_1, F_1)$, where $V_1 = \{p, q, r\}$, and F_1 and the corresponding NLP $\tau(N_1)$ are as follows.

$$
\begin{array}{lll}
F_1: & p(t+1) = q(t), & \\
 & q(t+1) = p(t) \wedge r(t), & \\
 & r(t+1) = \neg p(t). &
\end{array}
\qquad\qquad
\begin{array}{lll}
\tau(N_1): & p(t+1) \leftarrow q(t), & \\
 & q(t+1) \leftarrow p(t) \wedge r(t), & \\
 & r(t+1) \leftarrow \neg p(t). &
\end{array}
$$

The state transition diagram for N_1 is depicted in Fig. 1.[2]

Starting from the interpretation $J^0 = \{q(0), r(0)\}$, which means that q and r are true at time 0, its transitions with respect to the $T_{\tau(N_1)}$ operator are given as $J_1 = \{p(1), r(1)\}$, $J_2 = \{q(2)\}$, $J_3 = \{p(3), r(3)\}$, ..., which corresponds to the trajectory $qr \rightarrow pr \rightarrow q \rightarrow pr \rightarrow \ldots$ of N_1. Here $pr \rightarrow q \rightarrow pr$ is a cycle attractor (Fig. 1, below). N_1 has another, point attractor $r \rightarrow r$ (Fig. 1, above) whose basin of attraction is $\{pqr, pq, p, \epsilon, r\}$.

The second way to represent dynamics of Boolean networks is based on a recent work on the semantics of logic programming. Instead of using the above direct representation (3), we can consider another representation without the time argument. That is, we consider an NLP as a set of rules of the form (1). In [20], a Boolean network N is further translated to a propositional NLP $\pi(N)$ from $\tau(N)$ by deleting the time argument from every literal A^t appearing in $\tau(N)$. Then, we can simulate the trajectory of N from any state $S(0)$ also by the orbit of the interpretation $I^0 = \{v_i \in V \mid v_i(0) \text{ is true}\}$ with respect to the $T_{\pi(N)}$ operator, i.e., $I^{t+1} = T_{\pi(N)}(I^t)$ for $t \geq 0$. Moreover, we can characterize the attractors of N based on the *supported class semantics* [23] for $\pi(N)$.

A *supported class* of an NLP P [23] is a non-empty set $\mathcal{S}$ of Herbrand interpretations satisfying:

$$\mathcal{S} = \{T_P(I) \mid I \in \mathcal{S}\}. \tag{4}$$

Note that I is a supported model of P iff $\{I\}$ is a supported class of P. A supported class $\mathcal{S}$ of P is *strict* if no proper subset of $\mathcal{S}$ is a supported class of P. Alternatively, $\mathcal{S}$ is a strict supported class of P iff there is a directed cycle $I_1 \rightarrow I_2, \rightarrow \cdots \rightarrow I_k \rightarrow I_1$ $(k \geq 1)$ in the state transition diagram induced by T_P such that $\{I_1, I_2, \ldots, I_k\} = \mathcal{S}$ [23]. A strict supported class of $\pi(N)$ thus exactly characterizes an attractor of a Boolean network N.

[1] If no f_i is given to v_i, we assume the identity function for f_i, i.e., $v_i(t+1) = v_i(t)$.

[2] Each interpretation is concisely represented as a sequence of atoms instead of a set of atoms in examples, e.g., pq means $\{p, q\}$ and the empty string ϵ means $\emptyset$.

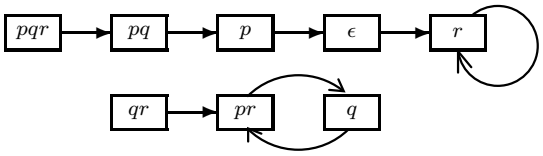

Fig. 1 The state transition diagram of N_1

Example 2 Consider the Boolean network N_1 in Example 1 again. The NLP

$$\pi(N_1): \quad \begin{array}{l} p \leftarrow q, \\ q \leftarrow p \wedge r, \\ r \leftarrow \neg p, \end{array}$$

is obtained from the first-order NLP $\tau(N_1)$ in Example 1 by removing the time argument from each literal. Notice that this logic program is not acyclic, since $\pi(N_1)$ has both positive and negative feedback loops: The positive loop appears between p and q, while the negative one exists in the dependency cycle to r through p. In this case, behavior of a corresponding Boolean network is not obvious.[3]

The state transition diagram induced by the $T_{\pi(N_1)}$ operator is the same as the diagram in Fig. 1. The orbit of pqr with respect to $T_{\pi(N_1)}$ becomes pqr, pq, p, ϵ, r, r, ... (Fig. 1, above), and the orbit of qr is qr, pr, q, pr, ... (Fig. 1, below). We here verify that there are two supported classes of $\pi(N_1)$, $\{\{r\}\}$ and $\{\{p, r\}, \{q\}\}$, which respectively correspond to the point attractor and the cycle attractor of N_1.

A further discussion on the selection of representation and the semantics for capturing dynamical systems in logic programs will be given in Section 7.3. In the following, we can use an NLP either with the time argument in the form of (3) or without the time argument in the usual form (1) for learning. To simplify the discussion, however, we will mainly use NLPs without the time argument in basic algorithms.

4 Learning from 1-Step Transitions

Now we consider learning from interpretation transition (LFIT). LFIT is an *anytime algorithm*, that is, whenever we process a set E of state transitions, we will guarantee that the result of learning is a logic program P which completely represents the dynamics of the transitions E so that a dynamical system is represented by P.

This section focuses on *learning from 1-step transitions* (**LF1T**) as LFIT. For learning, we assume that the Herbrand base $\mathcal{B}$ is finite.

Learning from 1-Step Transitions (LF1T)

Input: $E \subseteq 2^{\mathcal{B}} \times 2^{\mathcal{B}}$: (positive) examples/observations, an initial NLP P_0.
Output: An NLP P such that $J = T_P(I)$ holds for any $(I, J) \in E$.

[3] The reason why behavior becomes complex in the existence of feedbacks is biologically justified as follows. Each positive loop in a Boolean network is related to reinforcement and existence of multiple attractors, while each negative loop is the source of periodic oscillations involved in homeostasis [36].

In **LF1T**, a positive example is input as a one-step state transition, which is a pair of Herbrand interpretations.[4] We can also give a prior program P_0 before learning. The output of **LF1T** is an NLP which realizes all state transitions given in the input. Note that only one NLP is output by **LF1T**.

Here we show a bottom-up method to construct an NLP for **LF1T**. A bottom-up method generates hypotheses by *generalization* from the most specific clauses or examples until every positive example is covered. For two rules R_1, R_2 with the same head, R_1 *subsumes* R_2 if there is a substitution θ such that $b^+(R_1)\theta \subseteq b^+(R_2)$ and $b^-(R_1)\theta \subseteq b^-(R_2)$. In this case, R_1 is *more (or equally) general than* R_2, and R_2 is *less (or equally) general than* R_1. A rule R is the *least (general) generalization* [35] of R_1 and R_2, written as $R = lg(R_1, R_2)$, if R subsumes both R_1 and R_2 and is subsumed by any rule that subsumes both R_1 and R_2. According to Plotkin [35], the lg of two atoms $p(s_1, \ldots, s_n)$ and $q(t_1, \ldots, t_n)$ is undefined if $p \neq q$; and is $p(lg(s_1, t_1), \ldots, lg(s_n, t_n))$ if $p = q$ ($lg(s_i, t_i)$ is defined as in [35]). Then, $lg(R_1, R_2)$ is written as in Sakama [39]:

$$lg(h(R_1), h(R_2)) \leftarrow \bigwedge_{L \in b^+(R_1), K \in b^+(R_2)} lg(L, K) \ \wedge \bigwedge_{L \in b^-(R_1), K \in b^-(R_2)} \neg lg(L, K). \qquad (5)$$

The pseudo-code of **LF1T** is given as follows.

LF1T(E: pairs of Herbrand interpretations, P: an NLP)

1. If $E = \emptyset$ then output P and stop;
2. Pick $(I, J) \in E$, and put $E := E \setminus \{(I, J)\}$;
3. For each $A \in J$, let

$$R_A^I := \left(A \leftarrow \bigwedge_{B_i \in I} B_i \wedge \bigwedge_{C_j \in (\mathcal{B} \setminus I)} \neg C_j \right); \qquad (6)$$

4. If R_A^I is not subsumed by any rule in P, then $P := P \cup \{R_A^I\}$ and simplify P by generalizing some rules in P and removing all clauses subsumed by them;
5. Return to 1.

The **LF1T** algorithm can be used with or without an initial NLP P_0. Given the examples E only, **LF1T** is initially called by $\mathbf{LF1T}(E, \emptyset)$. If an initial NLP P_0 is given, $\mathbf{LF1T}(E, P_0)$ is called. **LF1T** firstly constructs the most specific rule R_A^I for each positive literal A appearing in $J = T_P(I)$ for each $(I, J) \in E$.[5] It is important here that *we do not construct any rule to make a literal false*. The rule R_A^I is then possibly generalized when another transition from E makes A true, which is computed by several generalization methods.

The first generalization method we consider is based on *resolution*. The resolution principle by Robinson [37] is well known as a deductive method, but its naïve use can be applied to a generalization method. In the following, for a literal l, $\overline{l}$ denotes the *complement* of l, i.e., when A is an atom, $\overline{A} = \neg A$ and $\overline{\neg A} = A$. We firstly consider a resolution between two ground rules as follows.

[4] A negative example (I, J) could be given if $J \neq T_P(I)$ is known and no positive example (I, K) such that $K = T_P(I)$ is known. Note that, once a positive example (I, K) is given, any pair (I, J) such that $J \neq K$ is regarded as a negative example.

[5] Based on the discussion in Section 3, we can alternatively consider a first-order expression of the form (3) as a rule in an output program P here. If we use a rule with the time argument, each R_A^I in **LF1T** becomes $A^{t+1} \leftarrow \bigwedge_{B_i \in I} {B_i}^t \wedge \bigwedge_{C_j \in (\mathcal{B} \setminus I)} \neg {C_j}^t$. In this case, generalization methods used in **LF1T** are essentially the same as those for the propositional expression; We apply each generalization just by keeping the time argument appearing in the body of each rule.

Definition 1 (naïve/ground resolution) Let R_1 and R_2 be two ground rules of the form (1), and l be a literal such that $h(R_1) = h(R_2)$, $l \in b(R_1)$ and $\overline{l} \in b(R_2)$. If $(b(R_2) \setminus \{\overline{l}\}) \subseteq (b(R_1) \setminus \{l\})$ then the *ground resolution of R_1 and R_2 (upon l)* is defined as

$$res(R_1, R_2) = \left(h(R_1) \leftarrow \bigwedge_{L_i \in b(R_1) \setminus \{l\}} L_i \right). \tag{7}$$

In particular, if $(b(R_2) \setminus \{\overline{l}\}) = (b(R_1) \setminus \{l\})$ then the ground resolution is called the *naïve resolution of R_1 and R_2 (upon l)*. In this particular case, the rules R_1 and R_2 are said to be *complementary* to each other *with respect to l*.

Both naïve resolution and ground resolution can be used as generalization methods of ground rules. For two ground rules R_1 and R_2, the naïve resolution $res(R_1, R_2)$ subsumes both R_1 and R_2, but the non-naïve ground resolution subsumes R_1 only.

Example 3 Suppose the three rules: $R_1 = (p \leftarrow q \wedge r)$, $R_2 = (p \leftarrow \neg q \wedge r)$, $R_3 = (p \leftarrow \neg q)$, and their resolvent: $res(R_1, R_2) = res(R_1, R_3) = (p \leftarrow r)$.

R_1 and R_2 are complementary with respect to q. Both R_1 and R_2 can be generalized by the naïve resolution of them because $res(R_1, R_2)$ subsumes both R_1 and R_2. On the other hand, the ground resolution $res(R_1, R_3)$ of R_1 and R_3 is equivalent to $res(R_1, R_2)$. However, $res(R_1, R_3)$ subsumes R_1 but does not subsume R_3.

Ground and naïve resolutions can be used to learn a ground NLP, and we will give the two corresponding versions of **LF1T** in Sections 4.1 and 4.2. These two algorithms are firstly used when there is no initial program, then an initial program is given as an input in Section 4.3. We also show how to learn non-ground NLPs in Section 4.4.

4.1 Generalization by Naïve Resolution

In our first implementation of **LF1T**, naïve resolution is used as a least generalization method. This method is particularly intuitive from the ILP viewpoint, since each generalization is performed based on a least generalization operator.

Proposition 1 *For two complementary ground rules R_1 and R_2, the naïve resolution of R_1 and R_2 is the least generalization of them, that is, $lg(R_1, R_2) = res(R_1, R_2)$.*

Proof Let R be $res(R_1, R_2)$. Since R subsumes both R_1 and R_2 in the case of naïve resolution, we here show that it is the least among such subsuming rules. Suppose that there is a rule R' such that (i) R' subsumes both R_1 and R_2, (ii) R' is subsumed by R, and (iii) R' does not subsume R. Since R and R' are ground, (ii) implies $b(R) \subseteq b(R')$, and then (iii) implies $b(R') \neq b(R)$. Then, there is a literal $l \in b(R')$ such that $l \notin b(R)$. By $l \in b(R')$ and $b(R') \subseteq b(R_1)$, $l \in R_1$ holds. But this only happens when l is resolved upon, i.e., $R = res(R_1, R_2) = (h(R_1) \leftarrow \bigwedge_{L_i \in b(R_1) \setminus \{l\}} L_i)$. However, by $b(R') \subseteq b(R_2)$, $l \in R_2$ holds too. Then l is not the literal resolved upon, a contradiction. □

When naïve resolution is used, we need an auxiliary set P_{old} of rules to globally store subsumed rules, which increases monotonically. P_{old} is set $\emptyset$ at first. When a generated rule is newly added at Step 4 in the pseudo-code of **LF1T**, we try to find a rule $R' \in P \cup P_{old}$ such that (a) $h(R') = h(R)$ and (b) $b(R)$ and $b(R')$ differ in the sign of only one literal l. If there is no such a rule R', then R is just added to P; otherwise, add R and R' to P_{old} then add $res(R, R')$ to P in a recursive call of Step 4.

The resulting algorithms for **LF1T** and **AddRule** are shown in Algorithms 1 and 2.

Algorithm 1 LF1T(E,P)

INPUT: a set E of pairs of Herbrand interpretations and an NLP P
OUTPUT: an NLP P

P_{old}: NLP
$P_{old} \leftarrow \emptyset$
while $E \neq \emptyset$ **do**
  Pick $(I, J) \in E$; $E := E \setminus \{(I, J)\}$
  for each $A \in J$ **do**
    $R^I_A := A \leftarrow \bigwedge_{B_i \in I} B_i \wedge \bigwedge_{C_j \in (\mathcal{B} \setminus I)} \neg C_j$
    AddRule(R^I_A, P, P_{old})
  end for
end while
return P

Algorithm 2 AddRule(R, P, P_{old}) (with naïve resolution)

INPUT : a rule R and two NLPs P and P_{old}

if R is subsumed by a rule of P **then**
  $P_{old} := P_{old} \cup \{R\}$
  return
end if
for each rule R_P of P subsumed by R **do**
  $P := P \setminus \{R_P\}$
  $P_{old} := P_{old} \cup \{R_P\}$
end for

$P := P \cup \{R\}$
// Check for generalizations
for each rule R' of $P \cup P_{old}$ with $h(R) = h(R')$ **do**
  for each $l \in b(R)$ such that $\overline{l} \in b(R')$ **do**
    if $b(R) \setminus \{l\} = b(R') \setminus \{\overline{l}\}$ **then**
      $P_{old} := P_{old} \cup \{R\}$
      $R^{lg} := h(R) \leftarrow \bigwedge_{L_i \in b(R) \setminus \{l\}} L_i$
      AddRule(R^{lg}, P, P_{old})
    end if
  end for
end for

Example 4 Consider the state transition in Fig. 1. By giving the state transitions step by step, the NLP $\pi(N_1) = \{\#11, \#14, \#19\}$ is obtained in Table 1, where $\#n$ is the rule ID.

We now examine the correctness of the **LF1T** algorithm in terms of its completeness and soundness. A program P is said to be *complete* for a set E of pairs of interpretations if $J = T_P(I)$ holds for any $(I, J) \in E$. On the other hand, P is *sound* for E if for any $(I, J) \in E$ and any $J' \in 2^{\mathcal{B}}$ such that $J' \neq J$, $J' \neq T_P(I)$ holds. A deterministic learning algorithm is *complete* (resp. *sound*) for E if its output program is complete (resp. sound) for E. We use the following subsumption relation between programs: Given two logic programs P_1 and P_2, P_1 *theory-subsumes* P_2 if for any rule $R \in P_2$, there is a rule $R' \in P_1$ such that R' subsumes R.

Theorem 1 (Completeness of LF1T with naïve resolution) *Given a set E of pairs of interpretations,* **LF1T** *with naïve resolution is complete for E.*

Table 1 Execution of **LF1T** in inferring $\pi(N_1)$ of Example 2

Step	$I \to J$	Operation	Rule	ID	P	P_{old}
1	$qr \to pr$	R_p^{qr}	$p \leftarrow \neg p \wedge q \wedge r$	1	1	$\emptyset$
		R_r^{qr}	$r \leftarrow \neg p \wedge q \wedge r$	2	1,2	
2	$pr \to q$	R_q^{pr}	$q \leftarrow p \wedge \neg q \wedge r$	3	1,2,3	
3	$q \to pr$	R_p^{q}	$p \leftarrow \neg p \wedge q \wedge \neg r$	4		
		$res(4,1)$	$p \leftarrow \neg p \wedge q$	5	2,3,5	+ 1,4
		R_r^{q}	$r \leftarrow \neg p \wedge q \wedge \neg r$	6		
		$res(6,2)$	$r \leftarrow \neg p \wedge q$	7	3,5,7	+ 2,6
4	$pqr \to pq$	R_p^{pqr}	$p \leftarrow p \wedge q \wedge r$	8		
		$res(8,1)$	$p \leftarrow q \wedge r$	9	3,5,7,9	+ 8
		R_q^{pqr}	$q \leftarrow p \wedge q \wedge r$	10		
		$res(10,3)$	$q \leftarrow p \wedge r$	11	5,7,9,11	+ 3,10
5	$pq \to p$	R_p^{pq}	$p \leftarrow p \wedge q \wedge \neg r$	12		
		$res(12,4)$	$p \leftarrow q \wedge \neg r$	13	5,7,9,11,13	+ 12
		$res(13,9)$	$p \leftarrow q$	14	7,11,14	+ 5,9,13
6	$p \to \epsilon$					
7	$\epsilon \to r$	R_r^{ϵ}	$r \leftarrow \neg p \wedge \neg q \wedge \neg r$	15		
		$res(15,6)$	$r \leftarrow \neg p \wedge \neg r$	16	7,11,14,16	+ 15
8	$r \to r$	R_r^{r}	$r \leftarrow \neg p \wedge \neg q \wedge r$	17		
		$res(17,15)$	$r \leftarrow \neg p \wedge \neg q$	18	7,11,14,16,18	+ 17
		$res(18,7)$	$r \leftarrow \neg p$	19	11,14,19	+ 7,16,18

Proof For any pair of interpretations $(I, J) \in E$, it is verified that the rule R_A^I determines the value of A in the next state of I correctly for any $A \in J$. On the other hand, for any atom $A \notin J$, the value of A in the next state of I becomes false by R_A^I and the T_P operator. Hence, the set of rules $P^* = \{R_A^I \mid (I, J) \in E, A \in J\}$ is complete for the transitions in E. Since a rule R derived by the naïve resolution of R_1 and R_2 subsumes R_1 and R_2 by Proposition 1, $P' = (P^* \setminus \{R_1, R_2\}) \cup \{R\}$ theory-subsumes P^*. Then, P' is also complete for E, since $T_{P'}$ and T_P agree with their transitions. Since the (theory-)subsumption relation is transitive, an output program P, which is obtained by repeatedly applying naïve resolutions, theory-subsumes P^*. Hence, P is complete for E. □

The implication of Theorem 1 is very important: *For any set of 1-step state transitions, we can construct an NLP that captures the dynamics in the transitions*. In other words, there is no (deterministic) state transition diagram that cannot be expressed in an NLP. It is also important to guarantee the soundness of the learning algorithm, that is, it never overgeneralizes any state transition rule. The soundness can be obtained from the completeness when the transition from any interpretation is deterministic like the assumption in this paper (that is why it is stated as a corollary), but we show a more precise proof for it.

Corollary 1 (Soundness of LF1T with naïve resolution) *Given a set E of pairs of interpretations,* **LF1T** *with naïve resolution is sound for E.*

Proof It is easy to see that the program P^* in the proof of Theorem 1 satisfies the soundness. Any naïve resolution $R = res(R_1, R_2)$ for any $R_1, R_2 \in P^*$ deletes only one literal l such that $l \in b(R_1)$ and $\overline{l} \in R_2$. Assume that $R_1 = R_A^{I_1}$ and $R_2 = R_A^{I_2}$ for some $(I_1, J_1) \in E$ and $(I_2, J_2) \in E$. Then, $b(R)$ is satisfied by any partial interpretation I' such that $I' = I_1 \cap I_2 = I_1 \setminus \{l\} = I_2 \setminus \{\overline{l}\}$. Considering total interpretations that are extensions of I', there are only two possibilities, i.e., I_1 and I_2. Since $A = h(R)$ belongs to both $J_1 = T_{P^*}(I_1)$ and $J_2 = T_{P^*}(I_2)$, it also belongs to $T_{P'}(I_1)$ and $T_{P'}(I_2)$, where $P' = (P^* \setminus \{R_1, R_2\}) \cup \{R\}$. Applying the same argument to all atoms in any $J = T_{P^*}(I)$ for any interpretation I, we

have $J = T_{P'}(I)$. This arguments can be further applied to all naïve resolutions, so that $T_P(I)$ is the same as $T_{P^*}(I)$ for the final NLP P. □

4.2 Generalization by Ground Resolution

Using naïve resolution, $P \cup P_{old}$ possibly contains all patterns of rules constructed from the Herbrand base $\mathcal{B}$ in their bodies. In our second implementation of **LF1T**, ground resolution is used as an alternative generalization method in **AddRule**. This replacement of resolution leads to a lot of computational gains, since we do not need P_{old} any more: Every generalization which can be found in P_{old} can be found in P by ground resolution.

Proposition 2 *All generalized rules obtained from $P \cup P_{old}$ by naïve resolution can be obtained using ground resolution on P.*

Proof Let $R_1 \in P$ and $R_2 \in P_{old}$ be ground complementary rules with respect to a literal $l \in b(R_1)$. Then, $h(R_1) = h(R_2)$, $\overline{l} \in b(R_2)$ and $(b(R_1) \setminus \{l\}) = (b(R_2) \setminus \{\overline{l}\})$ hold. Suppose that by naïve resolution, $R_3 = res(R_1, R_2)$ is put into P and that R_1 is put into P_{old} in **AddRule**. By $R_2 \in P_{old}$, there has been a rule R_4 in P such that R_4 subsumes R_2, that is, $b(R_4) \subseteq b(R_2)$. We can also assume that $\overline{l} \in b(R_4)$ because otherwise l has been resolved upon by the naïve resolution between R_2 and some rule in P and thus R_1 must have been put into P_{old}. Then, the rule $R_5 = res(R_1, R_4)$ is obtained by ground resolution, and $b(R_5) = (b(R_1) \setminus \{l\}) = (b(R_2) \setminus \{\overline{l}\})$. Hence R_5 is equivalent to R_3. □

Ground resolution can be used in place of naïve resolution to learn an NLP from traces of states transition. In this case, we can simplify Algorithm 1 by deleting Lines 3 and 4 and by replacing Line 9 with **AddRule**(R_A^I, P). Algorithm 3 describes the new **AddRule** which adds and simplify rules using ground resolution.

As in the case of naïve resolution, we can prove the correctness, i.e., the completeness and soundness of **LF1T** with ground resolution.

Theorem 2 (Completeness of LF1T with ground resolution) *Given a set E of pairs of interpretations,* **LF1T** *with ground resolution is complete for E.*

Proof As in the proof of Theorem 1, if a program P is complete for E, a program P' that theory-subsumes P is also complete for E. By Proposition 2, any rule produced by naïve resolution can be generated by ground resolution. Then, if P and P' are respectively obtained by naïve resolution and ground resolution, P' theory-subsumes P. Since P is complete for E by Theorem 1, P' is complete for E. □

Corollary 2 (Soundness of LF1T with ground resolution) *Given a set E of pairs of interpretations,* **LF1T** *with ground resolution is sound for E.*

Proof By Theorem 2, a program P output by **LF1T** with ground resolution is complete for E. Then, as in the proof of Corollary 1, P is shown to be sound for E. □

Example 5 Consider again the state transition in Fig. 1. Using ground resolution, the NLP $\pi(N_1) = \{\#11, \#14, \#19\}$ is obtained in Table 2.

Comparing Examples 2 and 5, all rules generated by naïve resolution are obtained by ground resolution too. By avoiding the use of P_{old}, however, we can reduce time and space

Algorithm 3 AddRule(R,P) (with ground resolution)

INPUT : a rule R and a NLP P

for each rule R_P of P **do**
  if R is subsumed by R_P **then**
    return
  end if
  if R subsumes R_P **then**
    $P := P \setminus \{R_P\}$
  else
    // Check for generalizations
    if $h(R) = h(R_P)$ **then**
      if $\exists l \in b(R)$ such that $\overline{l} \in b(R_P)$ **then**
        if $b(R) \setminus \{l\}$ is subsumed by $b(R_P) \setminus \{\overline{l}\}$ **then**
          $R^r := h(R) \leftarrow \bigwedge_{L_i \in b(R) \setminus \{l\}} L_i$
          AddRule(R^r, P)
          return
        end if
        if $b(R) \setminus \{l\}$ subsumes $b(R_P) \setminus \{\overline{l}\}$ **then**
          $R^r_P := h(R_P) \leftarrow \bigwedge_{L_i \in b(R_P) \setminus \{\overline{l}\}} L_i$
          AddRule(R^r_P, P)
          AddRule(R, P)
          return
        end if
      end if
    end if
  end if
end for
$P := P \cup \{R\}$

for learning. As the next theorem shows, ground resolution has much complexity gain compared with naïve resolution, when learning is done with the input of complete 1-step state transitions from all 2^n interpretations, where n is the size of the Herbrand base $\mathcal{B}$. In the propositional case, n is the number of propositional atoms, which correspond to the number of nodes in a Boolean network. We here assume that each operation of subsumption and resolution can be performed in time $O(1)$ by assuming a bit-vector data structure.

Theorem 3 *Using naïve version, the memory use of the* **LF1T** *algorithm is bounded by $O(n \cdot 3^n)$, and the time complexity of learning is bounded by $O(n^2 \cdot 9^n)$, where $n = |\mathcal{B}|$. On the other hand, with ground resolution, the memory use is bounded by $O(2^n)$, which is the maximum size of P, and the time complexity is bounded by $O(4^n)$.*

Proof In both P and P_{old}, the maximum size of the body of a rule is n. There are n possible heads and 3^n possible bodies for each rule: Each element of $\mathcal{B}$ can be either positive, negative or absent in the body of a rule. This means that both $|P|$ and $|P_{old}|$ are bounded by the size in $O(n \cdot 3^n)$. The memory use in the algorithm is thus $O(n \cdot 3^n)$. In practice, however, $|P|$ is less than or equal to $O(2^n)$ for the following reason. In the worst case, P contains only rules of size n; if P contains a rule with m literals ($m < n$), this rule subsumes 2^{n-m} rules which cannot appear in P. That is why we can consider only two possibilities for each literal, i.e., positive and negative occurrences of the literal (and no blank) to estimate the size $|P|$. Furthermore, P does not contain any pair of complementary rules, so that the complexity is further divided by n, that is, $|P|$ is bounded by $O(n \cdot 2^n/n) = O(2^n)$. But $|P_{old}|$ remains in the same complexity and the memory use of the algorithm in practice is still $O(n \cdot 3^n)$.

Table 2 Execution of **LF1T** with ground resolution in inferring $\pi(N_1)$ of Example 2

Step	$I \to J$	Operation	Rule	ID	P
1	$qr \to pr$	R_p^{qr}	$p \leftarrow \neg p \wedge q \wedge r$	1	1
		R_r^{qr}	$r \leftarrow \neg p \wedge q \wedge r$	2	1,2
2	$pr \to q$	R_q^{pr}	$q \leftarrow p \wedge \neg q \wedge r$	3	1,2,3
3	$q \to pr$	R_p^q	$p \leftarrow \neg p \wedge q \wedge \neg r$	4	
		$res(4,1)$	$p \leftarrow \neg p \wedge q$	5	2,3,5
		R_r^q	$r \leftarrow \neg p \wedge q \wedge \neg r$	6	
		$res(6,2)$	$r \leftarrow \neg p \wedge q$	7	3,5,7
4	$pqr \to pq$	R_p^{pqr}	$p \leftarrow p \wedge q \wedge r$	8	
		$res(8,5)$	$p \leftarrow q \wedge r$	9	3,5,7,9
		R_q^{pqr}	$q \leftarrow p \wedge q \wedge r$	10	
		$res(10,3)$	$q \leftarrow p \wedge r$	11	5,7,9,11
5	$pq \to p$	R_p^{pq}	$p \leftarrow p \wedge q \wedge \neg r$	12	
		$res(12,5)$	$p \leftarrow q \wedge \neg r$	13	
		$res(13,9)$	$p \leftarrow q$	14	7,11,14
6	$p \to \epsilon$				
7	$\epsilon \to r$	R_r^{ϵ}	$r \leftarrow \neg p \wedge \neg q \wedge \neg r$	15	
		$res(15,7)$	$r \leftarrow \neg p \wedge \neg r$	16	7,11,14,16
8	$r \to r$	R_r^r	$r \leftarrow \neg p \wedge \neg q \wedge r$	17	
		$res(17,7)$	$r \leftarrow \neg p \wedge r$	18	
		$res(18,16)$	$r \leftarrow \neg p$	19	11,14,19

In adding a rule to P in **AddRule** using naïve resolution, we have to compare it with all rules in $P \cup P_{old}$, then this operation has a complexity of $O(n \cdot 3^n)$. Hence, using naïve resolution, the complexity of **LF1T** is $O(\sum_{k=1}^{n \cdot 3^n} k)$, where k represent the number of rules in $P \cup P_{old}$, which increases during the process until it finally belongs to $O(n \cdot 3^n)$. Therefore, the complexity of learning with naïve version is $O(\sum_{k=1}^{n \cdot 3^n} k)$, which is then equal to $O(n^2 \cdot 3^{2n-1}) = O(n^2 \cdot 9^n)$. On the other hand, using ground resolution, the memory use of the **LF1T** algorithm is $O(2^n)$, which is the maximum size of P. The complexity of learning is then $O(\sum_{k=1}^{2^n} k)$, which is equal to $O((2^n(2^n+1))/2) = O(2^{2n-1}) = O(4^n)$. □

By Theorem 3, given the set E of complete state transitions, which has the size $O(2^n)$, the complexity of $\mathbf{LF1T}(E, \emptyset)$ with ground resolution is bounded by $O(|E|^2)$. On the other hand, the worst-case complexity of learning with naïve resolution is $O(n^2 \cdot |E|^{4.5})$. We will see the difference in experiments on learning biological networks in Section 6.1.

4.3 Background Theories and Incremental Learning

So far, we have not explicitly mentioned a *background theory* or a prior program that is given before learning. But this is easily handled in **LF1T**: If we are given a prior NLP P_0 as a background theory, we can just call $\mathbf{LF1T}(E, P_0)$.

Theorem 3 gives the upper bounds of the complexity of learning under the assumption that the set of complete state transitions is given as the input and no initial program is given. However, **LF1T** is an anytime algorithm and is hence complete for any incomplete set of state transitions with or without a prior program. Then, the next proposition shows the relationship between the size of inputs and the generality of programs learned by **LF1T** with either naïve or ground resolution.

Proposition 3 *Let E and E' be sets of state transitions such that $E \subseteq E'$. Let P be an NLP learned by $\mathbf{LF1T}(E, \emptyset)$, and P' be an NLP learned by $\mathbf{LF1T}(E' \setminus E, P)$. Then, P' theory-subsumes P.*

Proof For any rule R in P, either R remains in P' or is subsumed by a new rule R' obtained in P'. In either case, there is a rule in P' which subsumes R. Hence, P' theory-subsumes P. □

Since **LF1T** is complete for any input, any learned program has the same state transitions for any ordering of state transitions. Then, $P' = \mathbf{LF1T}(E' \setminus E, P)$ and $\mathbf{LF1T}(E', \emptyset)$ agree with their state transitions, which is $T_{P'}$. That is, **LF1T** can be performed in an *incremental* manner. Proposition 3 indicates that, the more examples are given, the more general programs are obtained. Actually, for any ground atom A, we will have a more general rule with head A than a rule in P or other new rules with head A in P'. As in the proof of Theorem 1, learning a rule with m body literals need 2^{n-m} examples for the naïve resolution method. We thus need more examples to get smaller rules in general.

4.4 Handling Non-Ground Cases

In Sections 4.1 and 4.2, we have assumed that no initial NLP is input to **LF1T**. In this case, only examples are given, which are transitions of interpretations that are ground. That is why we only needed ground resolution for generalization, but the resultant program is also ground. Here, we consider the case that an initial NLP P_0 is given as an input, where P_0 can contain variables. The next generalization operation is defined for any two non-ground rules, including the case that one rule is ground and the other is non-ground. Like ground resolution, a non-ground resolution $res(R_1, R_2)$ of two rules in the next definition produce a generalized rule that subsumes R_1.

Definition 2 (non-ground resolution) Let R_1 and R_2 be rules, and $l \in b(R_1)$ and $l' \in b(R_2)$ be literals such that $l = \overline{l'\theta}$ holds for some substitution θ. If $(b(R_2) \setminus \{l'\})\theta \subseteq (b(R_1) \setminus \{l\})$ holds, then $res(R_1, R_2) = (h(R_1) \leftarrow \bigwedge_{L_i \in b(R_1) \setminus \{l\}} L_i)$.

Note that non-ground resolution in Definition 2 is a special case of more general resolution in [37], which derives a rule with the body $((b(R_1) \cup b(R_2)) \setminus \{l', l\})\theta$ such that $l\theta = \overline{l'\theta}$ holds. That is, it is a special case that the relation $(b(R_2) \setminus \{l'\})\theta \subseteq (b(R_1) \setminus \{l\})$ holds. We will not consider this kind of general resolution in **LF1T**, since such a resolvent generalizes neither R_1 nor R_2.

As discussed in Section 3, a generalization not involving time can be performed by a standard ILP technique. For example, we can apply *anti-instantiation* (AI) as a generalization operator, which replaces a sub-term with a variable. We will see examples of learning non-ground rules in Section 6.2. Note that unrestricted applications of such generalization operators do not guarantee the soundness in general, so we need to check the consistency of generalized rules with the examples in applying those operators.

5 Variations

5.1 Learning from Basins of Attraction

Identification of an exact NLP using **LF1T** may require $2^{|\mathcal{B}|}$ examples, and this bound cannot be reduced in general [3]. In biological applications, however, this does not necessarily

mean that we need an exponential number of experimental GAP samples. Instead, we can observe changes of GAPs from time to time, and get trajectories from much fewer initial GAPs. Fortunately, any trajectory always reaches an attractor, so we can stop observing changes as soon as we encounter a previously observed GAP. This scenario derives us to design another LFIT framework to learn a Boolean network (or an NLP) from basins of attraction as follows.

Learning from Basins of Attraction (LFBA)

Input: A set $\mathcal{E}$ of orbits of interpretations (*).
Output: An NLP P such that, for every $\mathcal{I} \in \mathcal{E}$, any $I \in \mathcal{I}$ belongs to the basin of attraction of an attractor of P that is contained in $\mathcal{I}$.

In **LFBA**, an example $\mathcal{I} \in \mathcal{E}$ is given as a part of the basin of attraction of some attractor of the target NLP P. We here assume for the input (*) that each $\mathcal{I}$ contains the Herbrand interpretations belonging to the orbit of an initial interpretation $I_0 \in \mathcal{I}$ with respect to the T_P operator, and that every transition among $\mathcal{I}$ is completely known so that $\mathcal{I}$ can be written as a sequence $I_0 \to I_1 \to \cdots \to I_{k-1} \to J_0 \to \cdots \to J_{l-1} \to J_0 \to \cdots$, where $|\mathcal{I}| = k+l$ and $\{J_0, \cdots, J_{l-1}\}$ is an attractor. A set $\mathcal{E}$ of examples in **LFBA** has the property that two orbits $\mathcal{I}, \mathcal{J} \in \mathcal{E}$ reach the same attractor if and only if $\mathcal{I} \cap \mathcal{J} \neq \emptyset$ holds.

LFBA($\mathcal{E}$: orbits of Herbrand interpretations)
1. Put $P := \emptyset$;
2. If $\mathcal{E} = \emptyset$ then output P and stop;
3. Pick $\mathcal{I} \in \mathcal{E}$, and put $\mathcal{E} := \mathcal{E} \setminus \{\mathcal{I}\}$;
4. Put $E := \{(I, J) \mid I, J \in \mathcal{I},\ J \text{ is the next state of } I\}$;
5. $P := \mathbf{LF1T}(E, P)$; Return to 2.

The input size of learning an NLP by **LFBA** is bounded by the number of attractors in the given state transition diagrams. This is practically much lower than $2^{|\mathcal{B}|}$. However, in the worst case, there is a Boolean network which has an exponential number of attractors. For example, the NLP $\{(v_i \leftarrow v_i) \mid v_i \in \mathcal{B}\}$ has $2^{|\mathcal{B}|}$ point attractors.

Example 6 Consider again the state transition in Fig. 1. But this time, the input is given as $\mathcal{E} = \{\mathcal{I}_1, \mathcal{I}_2\}$, where $\mathcal{I}_1$ is the sequence: $qr \to pr \to q \to pr \to \cdots$, and $\mathcal{I}_2$ is the sequence: $pqr \to pq \to p \to \epsilon \to r \to r \to \cdots$. Put $E_1 = \{(qr, pr), (pr, q), (q, pr)\}$ and $E_2 = \{(pqr, pq), (pq, p), (p, \epsilon), (\epsilon, r), (r, r)\}$. Then, in **LFBA**, firstly $\mathbf{LF1T}(E_1, \emptyset)$ is called, and the resulting NLP P_1 is obtained as $P_1 = \{\#3, \#5, \#7\}$. Next, $\mathbf{LF1T}(E_2, P_1)$ is called and the NLP $\pi(N_1) = \{\#11, \#14, \#19\}$ is obtained.

5.2 Exogenous Events

Boolean networks are one of the simplest dynamical systems in the sense that all behaviors are deterministic and solely depend on the initial state and state transition rules. An important extension would be to introduce the notion of *exogenous events*. The importance of such exogenous events has been discussed in the literature [7], and they generally interfere normal transitions of states. Then, we need to distinguish those state transitions induced by the dynamical system itself and other state transitions caused by exogenous events. Learning such dynamics can be simply done in our framework by taking only those system's transitions as input examples and ignoring transitions perturbed or forced by external events.

Given the input state transitions, $I_0 \to I_1 \to I_2 \to \cdots \to I_{k-1} \to I_k \Rightarrow I_{k+1} \to I_{k+2} \to \cdots$, suppose that the transition from I_k to I_{k+1} (denoted by the double arrow $\Rightarrow$) is caused by some external event. Then, let $\mathcal{I}_1$ be $I_0 \to I_1 \to I_2 \to \ldots \to I_{k-1} \to I_k$ and $\mathcal{I}_2$ be $I_{k+1} \to I_{k+2} \to \ldots$. In this case, $\mathbf{LFBA}(\{\mathcal{I}_1, \mathcal{I}_2\})$ is applied by calling $\mathbf{LF1T}(E_1, \emptyset)$ first, and then calling $\mathbf{LF1T}(E_2, P_1)$, where $E_1 = \{(I_0, I_1), (I_1, I_2), \ldots, (I_{k-1}, I_k)\}$, $E_2 = \{(I_{k+1}, I_{k+2}), \ldots\}$, and P_1 is the result of $\mathbf{LF1T}(E_1, \emptyset)$.

5.3 Inductive Biases

Inductive biases can be incorporated into **LF1T** in various ways. For example, a prescribed set of literals that can affect the value of an atom A can be given for each $A \in \mathcal{B}$. In Boolean networks, we often know such "neighbor" literals, but may not know its exact Boolean function [4]. In such a case, we can focus on those input nodes $v_{i_1}, \ldots, v_{i_k} \in V$ of a node $v_i \in V$ in each interpretation I, and pick only those values of $v_{i_1}, \ldots, v_{i_k}$ in I when the body of $R^I_{v_i}$ is constructed in **LF1T**. In cellular automata, those neighborhood cells are already known for every cell, so this bias can be effectively used.

As another useful inductive bias, we can restrict the length of each learned rule R, i.e., $|b(R)| \leq k$ for some integer $k > 0$. When $|\mathcal{B}| = n$, the size of each I is also n, i.e., $|I| = n$ for any $I \in 2^{\mathcal{B}}$. When the length condition is $k < n$, there are two ways to meet this condition. The first method is to follow the algorithm of **LF1T** without restricting the length of the body of each produced rule, and wait until the length becomes less than k by resolution generalization. Once we have generated such a rule, the length condition is always satisfied by resolution generalization in Definitions 1 and 2. However, if a rule has more than k literals in the body at the end of learning, we need to shorten the body to meet the condition by selecting n literals from the body. This last selection must be done by keeping the consistency with the examples. The second method is more brave and constructs a rule R^I_A in **LF1T** for a literal A and an interpretation I by selecting only k values from I in constructing the body of R^I_A. This selection is nondeterministic, and may not guarantee the soundness by Corollaries 1 and 2. Then we need to check the consistency whenever a new example is processed, and need to backtrack to a selection point if a conflicting rule is produced. We will use both biases of neighbor literals and length conditions in experiments of Section 6.2.

5.4 Learning from Transitions of Partial Interpretations

In **LF1T**, we have assumed that an example is given as a set of interpretation transitions, in which each Herbrand interpretation is a subset of the Herbrand base $\mathcal{B}$. Such a (total) interpretation represents a complete assignment of truth values to all elements of $\mathcal{B}$. However, it is often the case that we can observe truth values for only a subset S of $\mathcal{B}$. Such an assignment $I \subseteq S \subseteq \mathcal{B}$ is called a *partial interpretation*, and those ground atoms not appearing in I is either assigned false for $S \setminus I$ or is missing in $\mathcal{B} \setminus S$ due to partial observability. We need to distinguish two cases to handle such missing values. In the first case, truth values of ground atoms in $\mathcal{B} \setminus S$ are just unknown. Then we can construct a rule R^I_A, but as in the second method for the length bias in Section 5.3, the soundness is not guaranteed, so that we need to check the consistency whenever a new example is input. In the second case, truth values of ground atoms in $\mathcal{B} \setminus S$ are "don't-care". Then we can safely construct a sound rule R^I_A, which does not need to be revised later for any input.

Table 3 Learning time of **LF1T** for Boolean networks up to 15 nodes

Name	# nodes	# × length of attractor	# rules (org./LFIT)	Naïve	Ground
Arabidopsis thalania	15	10×1	28 / 241	T.O.	13.825s
Budding yeast	12	7×1	54 / 54	6m01s	0.820s
Fission yeast	10	13×1	23 / 24	5.208s	0.068s
Mammalian cell	10	$1 \times 1,\ 1 \times 7$	22 / 22	5.756s	0.076s

An interesting application of the second case is "boosting", which runs **LF1T** again with those previously learned rules as input. In boosting, each rule R learned in the previous run is converted to a pair of partial interpretations $(b(R), h(R))$, and those atoms not appearing in $h(R)$ are just ignored (or are treated as 0) in the next state of each example. The boosting method can be used to simplify the learned rules by applying more (non-ground) resolutions, and further boostings can be performed again and again. Since resolution generalization in **LF1T** is performed only when the size of the resolvent is reduced, repeated boostings must terminate. The speed of convergence to the minimal reduced rules is generally much faster than performing complete resolutions in a learned program. In fact, it takes the number of resolutions in the factorial of the input size $|E|$ to perform the complete saturation strategy. With repeated boostings, however, we cannot remove all redundant rules in general. This method will be applied in constructing the rules for *Arabidopsis thalania* in Section 6.1.

6 Experiments

In this section, we evaluate our learning methods through experiments. We apply our LFIT algorithms to learn Boolean networks [24] in Section 6.1, and apply LFIT to identification of cellular automata [47] in Section 6.2.

6.1 Learning Boolean Networks

We here run our learning programs on some benchmarks of Boolean networks taken from Dubrova and Teslenko [15], which include those networks for control of flower morphogenesis in *Arabidopsis thaliana* [10], budding yeast cell cycle regulation [27], fission yeast cell cycle regulation [13] and mammalian cell cycle regulation [17]. However, since our problem setting for learning is different from that for computing attractors in [15], we needed to reproduce these inverse problems, which are made as follows. Firstly, we construct an NLP $\tau(N)$ from the Boolean function of a Boolean network N using the translation in Section 3, where each Boolean function is transformed to a DNF formula. Then, we get all possible 1-step state transitions of N from all $2^{\mathcal{B}}$ possible initial states I^0's by computing all stable models of $\tau(N) \cup I^0$ using the answer set solver `clasp`.[6] Finally, we use this set of state transitions to learn an NLP using our LFIT algorithms. Because a run of **LF1T** returns an NLP which can contain redundant rules, the original NLP P_{org} and the output NLP P_{LFIT} can be different, but remain equivalent with respect to state transition, that is, $T_{P_{org}}$ and $T_{P_{LFIT}}$ are identical functions.

Table 3 shows the time of a single **LF1T** run in learning a Boolean network for each problem in [15] on a processor Intel Core I7 (3610QM, 2.3GHz). The time limit is set to 1

[6] `http://potassco.sourceforge.net/`

hour for each experiment. We can see the good effect of using ground resolution in place of naïve resolution. The number of learned rules in each setting is also shown in Table 3, and is compared with the original literatures that present networks. Except *Arabidopsis thalania*, LFIT succeeds to reconstruct the same gene regulation rules as in [15] in the first run of **LF1T**. However, in *Arabidopsis thalania*, only 12 original rules are reproduced and the 16 other original rules are replaced with other learned 229 rules in the output of the first run of **LF1T**. Although those output rules are all minimal with respect to subsumption among them, some are subsumed by original rules. This is because, our resolution strategy is to perform resolution only when it produces a generalized rule, so other kinds of resolution, which was mentioned in Section 4.4 as general resolution, are not allowed. For example, from $R_1 = (p \leftarrow p \wedge q)$ and $R_2 = (p \leftarrow \neg q \wedge r)$, $R = (p \leftarrow p \wedge r)$ cannot be obtained in **LF1T**, since R subsumes neither R_1 nor R_2. Then, we applied boostings (Section 5.4) twice for *Arabidopsis thalania*, and obtained 76 rules in the first boosting, then got exactly the same 28 original rules in the second boosting. In constructing regulation rules of fission yeast, only one rule is additionally produced: $R_{15} = (x_5 \leftarrow \neg x_2 \wedge \neg x_4 \wedge x_5 \wedge x_6)$. This rule does not disappear with a boosting and the number of learned rules does not decrease from 24. Rules like R_{15} are not necessary to capture the whole transitions, but may give an alternative way to implement the dynamics. Hence, the same transition system can be realized in different ways. If this is considered as a redundancy, it might be useful for robustness of biological systems, but such analysis is beyond the scope of this paper.

In this experiment, the algorithm needs to analyze 2^n steps of transitions to learn an NLP, where n is the number of nodes in a Boolean network. That is why our implemented programs cannot handle networks with more than 20 nodes in the benchmark; computing all 1-step transitions takes too much time, since the grounding in the answer set solver cannot handle it. In other words, the input size with more than 2^{20} is too huge to be handled, so that we cannot even start learning. Such a limitation is acceptable in the ILP literature; for example, it has been stated that networks with 10 transitions and 10 nodes are reasonably large for learning [43]. Moreover, in real biological networks, we do not observe an exponential number of the whole state transitions from all possible initial states. Hence, anytime algorithms in this paper must be useful for such incomplete set of transitions, since learned programs are correct for any given partial set of state transitions.

6.2 Learning Cellular Automata

We here test to run **LF1T** with a background theory and inductive biases to learn transition rules of cellular automata. A *cellular automaton* (CA) [47] consists of a regular grid of *cells*, each of which has a finite number of possible states. The state of each cell changes synchronously in discrete time steps according to a local and identical *transition rule*. The state of a cell in the next time step is determined by its current state and the states of its surrounding cells (called *neighbors*). The collection of all cellular states in the grid at some time step is called a *configuration*. An *elementary CA* consists of a one-dimensional array of possibly infinite cells, and each cell has one of two possible states 0 (white, dead) or 1 (black, alive). A cell and its two adjacent cells form a neighbor of three cells, so there are $2^3 = 8$ possible patterns for neighbors. A transition rule describes for each pattern of a neighbor, whether the central cell will be 0 or 1 at the next time step.

In [1], Adamatzky poses the problem to identify a CA from an arbitrary pair of its configurations. We provide a solution to this problem using LFIT.

Table 4 Wolfram's Rule 110

Current pattern	111	110	101	100	011	010	001	000
New state for center cell	0	1	1	0	1	1	1	0

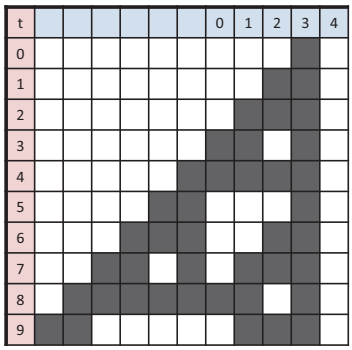

Fig. 2 State changes by Wolfram's Rule 110

Example 7 We here pick up one of the most famous elementary CAs, known as Wolfram's Rule 110 [47], whose transition rule is given in Table 4. In the table, the eight possible values of the three neighboring cells are shown in the first row, and the resulting value of the central cell in the next time step is shown in the second row. Rule 110 is known to be Turing-complete. The pattern generated by Rule 110 from the initial configuration with only one true cell (colored black) is depicted in Fig. 2. In the figure, time starts at 0 and patterns are shown until time 9. The column numbers are used later, and we here assign 3 to the column with the single black cell at time 0. We see that every cell at column 4 has the state 0 through transitions, since its neighbors always have the state 100 (assuming that the invisible column 5 has the state 0 at time 0).

We here reproduce the rules for Wolfram's Rule 110 from traces of configuration changes. Although this problem is rather simple, it illustrates how the whole system of LFIT with a background theory and inductive biases works to induce NLPs for CAs.

Originally, an infinite space is assumed for the CA with Rules 110. To deal with the CA in a finite space, two approximations can be considered:

1. **Limited frame**: Observes partially some set of cells. The problem in this setting is that, from the same configuration, different transitions can occur. For example, the configurations of cells $(1, 2, 3)$ at $t = 2$ and at $t = 4$ take the same values $(1, 1, 1)$ in Fig. 2, but the next states are $(1, 0, 1)$ at $t = 3$ and $(0, 0, 1)$ at $t = 5$. If the frame width is only 3, then we have two mutually inconsistent transitions from the same configuration. Hence, rules are not constructed for the two edge cells but are learned only for the internal cells.
2. **Torus world**: Assumes that there is no end in the shape of circle, doughnut or sphere, which can be constructed by chaining one edge cell with the other in one-dimensional cell patterns. Fig. 3 shows a torus world of size 4 and the state transitions by Rule 110 with the initial configuration $(1, 2, 3, 4) = (0, 0, 1, 0)$. The columns numbered (4) and (1) are thus identical to columns 4 and 1, respectively. Note that the configurations reach to the attractor, $(1, 1, 1, 0) \rightarrow (1, 0, 1, 1) \rightarrow (1, 1, 1, 0)$.

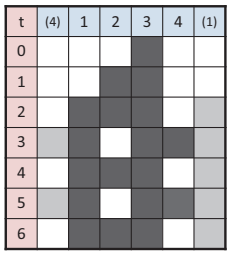

Fig. 3 State changes by Wolfram's Rule 110 in Torus world

Due to these approximations, the number of possible state transitions can be made smaller in the case of elementary CAs like Fig. 3. Our learning framework can handle both limited frames and torus worlds by considering adequate state transitions representation as input. For example, to represent a torus world of size 4, a configuration is represented by a vector with 6 elements $(0, 1, 2, 3, 4, 5)$: $1, 2, 3, 4$ respectively represent their values in the corresponding cells, and $0, 5$ respectively represent the values of cells 4 and 1 (colored gray when the value is 1). This last information can be represented in a background theory as the two rules with the time argument:

$$c(0,t) \leftarrow c(4,t),$$
$$c(5,t) \leftarrow c(1,t),$$

where $c(x)$ represents a cell x and $c(x,t)$ is its state at time step t. Unfortunately, these two rules do not have corresponding rules without the time argument, since the head literals refer the time step t instead of $t+1$. Hence, simple removal of the time argument from the both sides changes the dynamic meaning of the NLP in application of the T_P operator that infers about the next time step. Then, without the time argument, we should copy the rules for $c(4)$ to those for $c(0)$, and copy the rules for $c(1)$ to those for $c(5)$.

Now we use non-ground resolution and consider the following two biases (Section 5.3):

- **Bias I**: The body of each rule contains at most n neighbor literals.
- **Bias II**: The rules are universal for every time step and for any position. This means that the same states of the neighbor cells always implies the same state in the center cell at the next time step.

Combining these two biases, we can adapt **LF1T** to learn dynamics of CAs. Using Bias I, the rule construction process only considers n literals (here $n = 3$) in the neighbors of the cell in the body of a rule. With Bias I, ground resolution is not sufficient to compare non-ground rules with ground rules, for that we need non-ground resolution. We apply anti-instantiation (AI) for getting universal rules with Bias II, whenever a newly added rule R_A^I is not subsumed by any rule in the current program. We can guarantee the soundness of this generalization under Bias II. However, without Bias I, we cannot determine the body literals for construction of each universal rule, so that we must examine the effects from non-neighbor cells too.

Table 5 LF1T algorithm with Bias I on Rule 110 in Torus world

Step	$I \rightarrow J$	Operation	Rule	ID	P
1	$000100 \rightarrow 001100$	$R^{001}_{c(2)}$	$c(2) \leftarrow \neg c(1) \wedge \neg c(2) \wedge c(3)$	1	1
		$R^{010}_{c(3)}$	$c(3) \leftarrow \neg c(2) \wedge c(3) \wedge \neg c(4)$	2	1,2
2	$001100 \rightarrow 011101$	$R^{001}_{c(1)}$	$c(1) \leftarrow \neg c(0) \wedge \neg c(1) \wedge c(2)$	3	
		$lg(3,1)$	$c(x) \leftarrow \neg c(x-1) \wedge \neg c(x) \wedge c(x+1)$	4	2,4
		$R^{011}_{c(2)}$	$c(2) \leftarrow \neg c(1) \wedge c(2) \wedge c(3)$	5	
		$res(5,4)$	$c(2) \leftarrow \neg c(1) \wedge c(3)$	6	2,4,6
		$R^{110}_{c(3)}$	$c(3) \leftarrow c(2) \wedge c(3) \wedge \neg c(4)$	7	
		$res(7,2)$	$c(3) \leftarrow c(3) \wedge \neg c(4)$	8	4,6,8
3	$011101 \rightarrow 110111$	$R^{011}_{c(1)}$	$c(1) \leftarrow \neg c(0) \wedge c(1) \wedge c(2)$	9	
		$lg(9,6)$	$c(x) \leftarrow \neg c(x-1) \wedge c(x) \wedge c(x+1)$	10	
		$lg(10,4)$	$c(x) \leftarrow \neg c(x-1) \wedge c(x+1)$	11	8,11
		$R^{110}_{c(3)}$	$c(3) \leftarrow c(2) \wedge c(3) \wedge \neg c(4)$	12	
		$R^{101}_{c(4)}$	$c(4) \leftarrow c(3) \wedge \neg c(4) \wedge c(5)$	13	
		$res(13,11)$	$c(4) \leftarrow \neg c(4) \wedge c(5)$	14	8,11,14
4	$110111 \rightarrow 011101$	$R^{110}_{c(1)}$	$c(1) \leftarrow c(0) \wedge c(1) \wedge \neg c(2)$	15	
		$lg(15,8)$	$c(x) \leftarrow c(x-1) \wedge c(x) \wedge \neg c(x+1)$	16	8,11,14,16
		$R^{101}_{c(2)}$	$c(2) \leftarrow c(1) \wedge \neg c(2) \wedge c(3)$	17	
		$lg(17,14)$	$c(x) \leftarrow c(x-1) \wedge \neg c(x) \wedge c(x+1)$	18	
		$res(18,11)$	$c(x) \leftarrow \neg c(x) \wedge c(x+1)$	19	8,11,16,19
		$R^{011}_{c(3)}$	$c(3) \leftarrow \neg c(2) \wedge c(3) \wedge c(4)$	20	
		$res(20,19)$	$c(3) \leftarrow c(3) \wedge c(4)$	21	
		$res(21,19)$	$c(3) \leftarrow c(4)$	22	11,16,19,22
		$res(8,22)$	$c(3) \leftarrow c(3)$	23	11,16,19,22,23

Using Bias I only, **LF1T** in a limited frame of width 4 learns the following rules for Wolfram's Rule 110:

$$\begin{array}{l} c(2) \leftarrow \neg c(2) \wedge c(3), \\ c(3) \leftarrow \neg c(2) \wedge c(3), \\ c(3) \leftarrow c(3) \wedge \neg c(4), \\ c(x) \leftarrow \neg c(x-1) \wedge c(x+1), \\ c(x) \leftarrow c(x-1) \wedge c(x) \wedge \neg c(x+1). \end{array} \tag{8}$$

Instead, when we use a torus world of length 4 for Wolfram's Rule 110 in **LF1T** with Bias I only, Table 5 shows the learning process[7] and the following NLP is obtained:

$$\begin{array}{l} c(3) \leftarrow c(3), \\ c(3) \leftarrow c(4), \\ c(x) \leftarrow \neg c(x) \wedge c(x+1), \\ c(x) \leftarrow \neg c(x-1) \wedge c(x+1), \\ c(x) \leftarrow c(x-1) \wedge c(x) \wedge \neg c(x+1). \end{array} \tag{9}$$

Both programs (8) and (9) are quite different from the original rules in Table 4. On the other hand, if we use Biases I and II in either a limited frame of width 4 or a torus world of length 4, we get the following NLP (the process is in Table 5), which are equivalent to the original

[7] In Tables 5 and 6, interpretations I and J are represented as configurations, that is, $c(i) \in I$ iff $c(i)$ is true. Operation $lg(R_1, R_2)$ takes the least generalization of R_1 and R_2 with the same pattern, which generalizes the common terms in R_1 and R_2 into variables, and $ai(R)$ takes the anti-instantiation of R.

Table 6 LF1T algorithm with Biases I and II on Rule 110 in Torus world

Step	$I \to J$	Operation	Rule	ID	P
1	$000100 \to 001100$	$R^{001}_{c(2)}$	$c(2) \leftarrow \neg c(1) \wedge \neg c(2) \wedge c(3)$	1	
		$ai(1)$	$c(x) \leftarrow \neg c(x-1) \wedge \neg c(x) \wedge c(x+1)$	2	2
		$R^{010}_{c(3)}$	$c(3) \leftarrow \neg c(2) \wedge c(3) \wedge \neg c(4)$	3	
		$ai(3)$	$c(x) \leftarrow \neg c(x-1) \wedge c(x) \wedge \neg c(x+1)$	4	2,4
2	$001100 \to 011101$	$R^{001}_{c(1)}$	$c(1) \leftarrow \neg c(0) \wedge \neg c(1) \wedge c(2)$	5	
		$R^{011}_{c(2)}$	$c(2) \leftarrow \neg c(1) \wedge c(2) \wedge c(3)$	6	
		$ai(6)$	$c(x) \leftarrow \neg c(x-1) \wedge c(x) \wedge c(x+1)$	7	
		$res(7,2)$	$c(x) \leftarrow \neg c(x-1) \wedge c(x+1)$	8	8,4
		$res(7,4)$	$c(x) \leftarrow \neg c(x-1) \wedge c(x)$	9	8,9
		$R^{110}_{c(3)}$	$c(3) \leftarrow c(2) \wedge c(3) \wedge \neg c(4)$	10	
		$ai(10)$	$c(x) \leftarrow \neg c(x-1) \wedge c(x) \wedge \neg c(x+1)$	11	
		$res(11,9)$	$c(x) \leftarrow c(x) \wedge \neg c(x+1)$	12	8,9,12
3	$011101 \to 110111$	$R^{011}_{c(1)}$	$c(1) \leftarrow \neg c(0) \wedge c(1) \wedge c(2)$	13	
		$R^{110}_{c(3)}$	$c(3) \leftarrow c(2) \wedge c(3) \wedge \neg c(4)$	14	
		$R^{101}_{c(4)}$	$c(4) \leftarrow c(3) \wedge \neg c(4) \wedge c(5)$	15	
		$ai(15)$	$c(x) \leftarrow c(x-1) \wedge \neg c(x) \wedge c(x+1)$	16	
		$res(16,8)$	$c(x) \leftarrow \neg c(x) \wedge c(x+1)$	17	8,9,12,17
4	$110111 \to 011101$	$R^{110}_{c(1)}$	$c(1) \leftarrow c(0) \wedge c(1) \wedge \neg c(2)$	18	
		$R^{101}_{c(2)}$	$c(2) \leftarrow c(1) \wedge \neg c(2) \wedge c(3)$	19	
		$R^{011}_{c(3)}$	$c(3) \leftarrow \neg c(2) \wedge c(3) \wedge c(4)$	20	8,9,12,17

transition rule in Table 4:

$$\begin{aligned} &c(x) \leftarrow c(x) \wedge \neg c(x+1),\\ &c(x) \leftarrow \neg c(x-1) \wedge c(x),\\ &c(x) \leftarrow \neg c(x-1) \wedge c(x+1),\\ &c(x) \leftarrow \neg c(x) \wedge c(x+1). \end{aligned} \tag{10}$$

In learning an NLP for Rule 110 with Biases I and II, we get interesting generalizations. The NLP obtained from the trace of Rule 110 with **LF1T** becomes more compact in 4 rules, whereas the original transition rule representing the dynamics of this CA in Table 4 consists of 5 rules. However, there still exists a redundancy here; we can omit either the second or the third rule from (10). As discussed in Section 6.1, **LF1T** does not currently provide a method to remove irredundant rules from the learned rules.

7 Related Work

7.1 Learning from Interpretations

As stated in Section 1, *learning from interpretations* (LFI) [14] has been an ILP framework to produce a program from its interpretations. LFI considers examples simply as single interpretations that are supposed to be models of an output program, hence is different from LFIT, which takes pairs of interpretations as its input. We actually see that LFI is a special case of LFIT. That is, LFI can be constructed from LFIT as follows. Since $I \in 2^{\mathcal{B}}$ is a model of P iff $T_P(I) \subseteq I$, we can classify each example $(I, J) \in 2^{\mathcal{B}} \times 2^{\mathcal{B}}$ for LFIT into a positive example for LFI if $J \subseteq I$ or a negative example for LFI otherwise. Note that information of J is only used to check if I is a model or not in this conversion.

Then, there is still a difference between the above LFI and the conventional LFI by De Raedt [14]. The setting for De Raedt's LFI learns a *clausal theory*, i.e., a set of clauses, instead of an NLP that is a set of rules of the form (1). A clause is simply a disjunction of literals, while a positive literal and a negative literal in the body are clearly distinguished in a rule of an NLP. Other than this syntactical difference, the algorithm of conventional LFI can be used to construct a clausal theory from our input.

More generally, learning Boolean functions in the field of *computational learning theory* [25] is different from LFIT, since LFIT learns dynamics of systems as a set of Boolean functions appearing in Boolean networks, while the conventional learning setting is not involved in dynamics and often learns single Boolean functions. Similar to LFI, computational learning theories usually do not learn dynamics of systems in general.

7.2 Learning Action Theories

Learning action theories [31,33,21,45,12,38] can be considered to share the common goals with LFIT on learning dynamics. Moyle [31] uses an ILP technique to learn a causal theory based on event calculus [26], given examples of input-output relations. Otero [33] uses logic programs based on situation calculus [30], and considers causal theories represented in logic programs. Inoue *et al.* [21] induce causal theories represented in an action language given an incomplete action description and observations. Tran and Baral [45] define an action language which formalizes causal, trigger and inhibition rules to model signaling networks, and learn an action description in this language, given a candidate set of possible abducible rules. Active learning of action models is proposed by Rodrigues *et al.* [38] in a STRIPS-like language. Probabilistic logic programs to maximize the probabilities of observations are learned by Corapi *et al.* [12] by employing parameter estimation to find the probabilities associated with each atom and rule.

Works on learning action theories suppose applications to robotics and bioinformatics. In many action theories, one action is assumed to be performed at a time, so its learning task becomes sequential for each example sequence. In LFIT, on the other hand, every rule is fired as long as its body is satisfied and update is synchronously performed at every ground atom. Moreover, the goal of learning action theories is not exactly the same as that of LFIT. In particular, LFIT can learn dynamics of systems with *positive and negative feedbacks*, which has not been considered much in the literature.

Džeroski *et al.* [16]'s *relational reinforcement learning* (RRL) is a learning technique that combines reinforcement learning with ILP. As in (non-relational) reinforcement learning, RRL can take into account feedbacks from the learning process as rewards: Each time an observation is received, an action is chosen so that the state is changed with the reward associated. The goal of RRL is then to find a suitable sequence of transitions that maximize rewards. The merit to use ILP in RRL is to have a more expressive representation in states, actions and Q-functions. As the motivation of RRL is different from that of LFIT, our goal is not to find an optimal strategy for state transition but to learn the system's dynamics itself. As for the treatment of positive and negative feedbacks, LFIT learns how such feedbacks can be represented by logic programs.

7.3 Learning Nonmonotonic Programs

Learning NLPs has been considered in ILP, e.g., [39], but most approaches do not take the LFI setting. The LFI setting in learning NLPs is seen in Sakama [40]. Our learning framework is different from these previous works [39,40]. From the application point of view, NLPs are often used in planning and robotics domain, and hence the difference between previous work on learning action theories and LFIT is inherited to the comparison between previous setting of learning NLPs and LFIT. From the semantical viewpoint, there is an additional important difference: Previous work on learning NLPs is usually based on the stable model semantics [19], but LFIT learns NLPs under the supported model (or supported set) semantics [23]. Here we discuss practical differences between these two semantics.

The merit of the stable model semantics is that we can use state-of-the-art answer set solvers for computation of stable models. In [41], transition rules of CAs are represented in first-order NLPs, which consist of rules of the form (3) with the time argument. In this case, each NLP with the time argument becomes acyclic so the supported models and stable models coincide, and thus we can use answer set solvers for simulation of a CA. However, each answer set becomes infinite unless a time bound is set. On the other hand, the merit of the supported model semantics is that we can omit the time argument from a program and make it simpler. As discussed in Section 3, Boolean networks can be represented in propositional NLPs [20], but still we can simulate state transition by watching the orbits of the T_P operator. More importantly, attractors can be directly obtained with the supported model or the supported set semantics. This is not possible using the stable models of NLPs (without the time argument), since they ignore all positive feedback loops in the dynamics [20]. The supported models of an NLP can also be obtained as the models of Clark's completion of the program using modern SAT solvers. If we use answer set solvers for an NLP with the time argument, we can simulate the dynamics of the corresponding Boolean networks, but need to analyze each answer set to know when the same state is encountered twice by tracing the orbit from time to time.

7.4 Learning Boolean Networks and Cellular Automata

Learning the dynamics of Boolean networks has been considered in Bioinformatics. Liang *et al.* [28] proposed the REVEAL algorithm, which uses mutual information in information theory as a measure of interrelationships. In REVEAL, the maximum number of arguments of each Boolean function is assumed to deal with exponential growth of computational time. Akutsu *et al.* [3] analyze the problem of identifying a genetic network from the data obtained by multiple gene disruptions and overexpressions with respect to the number of experiments. They show algorithms for identifying the underlying genetic network by such experiments, but their network model is a static Boolean network model in which expression levels of genes are statically determined, and is hence different from the standard Boolean network in which expression levels of genes change synchronously. Pal *et al.* [34] constructs Boolean networks from a partial description of state transitions. This method is considered as a method to complete missing transitions in the state transition table. However, Boolean functions are not constructed for each node in [34]. Compared with these studies, our learning method is a complete algorithm to learn a set of logical state transition rules for a Boolean network. As in [34], we can also deal with partial transitions (Section 5.4), but will not identify or enumerate all possible complete transitions. Akutsu *et al.* [4] guess unknown Boolean functions of a Boolean network whose network topology is known. This

corresponds to learning Boolean networks with the bias of neighbor nodes (Section 5.3). In [4], only acyclic networks are considered, and the main focus is a computational analysis of such problems. Notably, all these previous works do not use ILP techniques.

In ILP, Srinivasan and Bain [43] present a framework to learn Petri nets from state transitions. Petri nets can handle quantities of entities but their update schemes are different from those of Boolean networks. In [43], a hierarchical Petri net can be obtained by iterative applications of their algorithm, but it is not possible to obtain networks with positive and negative feedback cycles. In fact, cyclic dependencies have been generally hard to be learned in ILP methods. Tamaddoni-Nezhad *et al.* [44] combine abduction and induction to learn rules of concentration changes of a metabolite caused by changes in other metabolites in a metabolic pathway. This method gives an empirical way to learn some causal effects, but its application domain does not deal with dynamical effects of feedbacks, and a learned program does not describe complete transitions of the dynamical system. Inoue *et al.* [22] complete causal networks by meta-level abduction. A biological network can be constructed with this method for an incomplete structure, but the abductive method does not consider dynamical behavior of the network and cannot deal with negative feedbacks.

In cellular automata (CAs), constructing transition rules from given configurations is known as the *identification problem*. Adamatzky [1] provides algorithms for identifying different classes of CAs, and analyzes computational complexities of those algorithms. Several algorithms are also proposed in [2]. To the best of our knowledge, however, there is no algorithm which uses ILP techniques for identifying CA rules.

8 Conclusion and Future Work

Learning complex networks becomes more and more important, but it is hard to infer rules of systems dynamics due to presence of positive and negative feedbacks. We here firstly tackled the induction problem of such dynamical systems in terms of NLP learning from synchronous state transitions. The proposed algorithm **LF1T** has the following properties:

- Given any state transition diagram, which is either complete or partial, we can learn an NLP that exactly captures the system dynamics.
- Learning is performed only from positive examples, and produces NLPs that consist only of rules to make literals true.
- Generalization on state transition rules is done by resolution, in which each rule can only be replaced by a general rule. As a result, an output NLP is as minimal as possible with respect to the size of each rule, but may contain redundant rules.

We have also shown how to incorporate background knowledge and inductive biases, and have applied the framework to learning transition rules of Boolean networks and cellular automata. The results are promising, and implemented programs can be useful for designing the state transition rules of dynamical systems from a specification of desired or non-desired state transition diagrams. For instance, a system can be considered to be *robust* if it is tolerant to a perturbation (or an exogenous event, see Section 5.2) which interferes normal state transition. Such a transition diagram could be designed as a tree shape, in which its root node corresponds to an attractor, so that any forced state change is eventually recovered to reach to the attractor [27]. Then we can do reverse engineering to get the corresponding state transition rules for the Boolean network.

A promising optimization of the implementation will be to use *binary decision diagrams* (BDDs) [9] to represent the rules of an NLP P in a compressed way. This data structure will

be more efficient with regard to memory and search spaces. With such representation, we can divide the complexity of learning step transitions by n: For one transition the algorithm learn n rules with the same body, if we use heads as leaves of the BDD, bodies of these rules will be learned and represented in only one multi-terminal BDD.

More complex schemes such as asynchronous and probabilistic updates [42, 18] do not obey transition by the T_P operator. Not only Boolean but multiple-state dynamical systems have been considered in the literature, so learning such systems is a future work. Cellular automata with asynchronous or probabilistic updates and their identification methods have also been proposed [1]. Those nondeterministic transition systems are more tolerant to noise, so can be expected to be applied to real-world dynamical systems, but have not yet been sufficiently connected to existing work on symbolic inference systems. Learning such dynamical networks by extending the algorithms in this paper is thus an important future work. Probabilistic logic learning would be useful for such learning tasks.

References

1. Adamatzky, A., *Identification of Cellular Automata*, CRC Press (1994)
2. Adamatzky, A. (ed.), Identification of Cellular Automata, Special Issue of *Journal of Cellular Automata*, 2(1) (2007)
3. Akutsu, T., Kuhara, S., Maruyama, O. and Miyano, S., Identification of genetic networks by strategic gene disruptions and gene overexpressions under a Boolean model, *Theoretical Computer Science*, 298:235–251 (2003)
4. Akutsu, T., Tamura, T. and Horimoto, K., Completing networks using observed data, *Proceedings of ALT, '09*, LNAI 5809, pp.126–140, Springer (2009)
5. Apt, K.R. and Bezem, M., Acyclic programs, *New Generation Computing*, 9:335–363 (1991)
6. Apt, K.R., Blair, H.A. and Walker, A., Towards a theory of declarative knowledge, in: Minker, J. (ed.), *Foundations of Deductive Databases and Logic Programming*, pp.89–148, Morgan Kaufmann (1988)
7. Baral, C., Eiter, T., Bjäreland, M. and Nakamura, M., Maintenance goals of agents in a dynamic environment: Formulation and policy construction, *Artificial Intelligence*, 172(12/13):1429–1469 (2008)
8. Blair, H.A., Chidella, J., Dushin, F., Ferry, A. and Humenn, P., A continuum of discrete systems, *Annals of Mathematics and Artificial Intelligence*, 21:153–186 (1997)
9. Bryant, R., Graph-based algorithms for Boolean function manipulation, *IEEE Trans. Computers*, 35(8):677–691 (1986)
10. Chaos, A., Aldana, M., Espinosa-Soto, C., Ponce de Léon, B., Arroyo, A.G. and Alvarez-Buylla, E.R., From genes to flower patterns and evolution: Dynamic models of gene regulatory networks, *Journal of Plant Growth Regulation*, 25(4):278–289 (2006)
11. Clark, K.L., Negation as failure, in: Gallaire, H. and Minker, J. (eds.), *Logic and Data Bases*, pp.119–140, Plenum Press, New York (1978)
12. Corapi, D., Sykes, D., Inoue, K. and Russo, A., Probabilistic rule learning in nonmonotonic domains, in: *Computational Logic in Multi-Agent Systems: Proceedings of the 12th International Workshop* (CLIMA-XII), LNAI 6814, pp.243–258, Springer (2011)
13. Davidich, M.I. and Bornholdt, S., Boolean network model predicts cell cycle sequence of fission yeast, *PLoS ONE*, 3(2), e1672 (2008)
14. De Raedt, L., Logical settings for concept-learning, *Artificial Intelligence*, 95:187–201 (1997)
15. Dubrova, E. and Teslenko, M., A SAT-based algorithm for finding attractors in synchronous Boolean networks, *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(5):1393–1399 (2011)
16. Džeroski, S., De Raedt, L. and Driessens, K., Relational reinforcement learning, *Machine Learning*, 43:7–52 (2001)
17. Fauré, A., Naldi, A., Chaouiya, C. and Thieffry, D., Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle, *Bioinformatics*, 22(14):e124–e131 (2006)
18. Garg, A., Di Cara, A., Xenarios, I., Mendoza, L and De Micheli, G., Synchronous versus asynchronous modeling of gene regulatory networks, *Bioinformatics*, 24(17):1917–1925 (2008)
19. Gelfond, M. and Lifschitz, V., The stable model semantics for logic programming, in: *Proceedings of ICLP '88*, pp.1070–1080, MIT Press (1988)
20. Inoue, K., Logic programming for Boolean networks, in: *Proceedings of IJCAI-11*, pp.924–930, AAAI Press (2011)

21. Inoue, K., Bando, H. and Nabeshima, H., Inducing causal laws by regular inference, in: *Proceedings of ILP '05*, LNAI 3625, pp.154–171, Springer (2005)
22. Inoue, K., Doncescu, A. and Nabeshima, H., Completing causal networks by meta-level abduction, *Machine Learning*, to appear (2013)
23. Inoue, K. and Sakama, C., Oscillating behavior of logic programs, in: Erdem, E., Lee, J., Lierler, Y., Pearce, D. (eds.), *Correct Reasoning—Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, LNAI 7265, pp.345–362, Springer (2012)
24. Kauffman, S.A., *The Origins of Order: Self-Organization and Selection in Evolution*, Oxford University Press (1993)
25. Kearns, M.J. and Vazirani, U.V., *An Introduction to Computational Learning Theory*, MIT Press (1994)
26. Kowalski, R.A. and Sergot, M.J., A logic-based calculus of events, *New Generation Computing*, 4(1):67–95 (1986)
27. Li, F., Long, T., Lu, Y., Ouyang, Q. and Tang, C., The yeast cell-cycle network is robustly designed, *PNAS*, 101(14):4781–4786 (2004)
28. Liang, S., Fuhrman, S. and Somogyi, R., REVEAL, a general reverse engineering algorithm for inference of genetic network architectures, in: *Pacific Symposium on Biocomputing*, 3:18–29 (1998)
29. Marek, W. and Subrahmanian, V.S., The relationship between stable, supported, default and autoepistemic semantics for general logic programs, *Theoretical Computer Science*, 103(2):365–386 (1992)
30. McCarthy, J. and Hayes, P.J., Some philosophical problems from the standpoint of artificial intelligence, in: B. Meltzer and D. Michie (eds.), *Machine Intelligence*, Vol.4, pp.463–502, Edinburgh University Press (1969)
31. Moyle, S., Using theory completion to learn a robot navigation control programs, in: *Proceedings of ILP '02*, LNAI 2583, pp.182–197, Springer (2003)
32. Muggleton, S.H., De Raedt, L., Poole, D., Bratko, I., Flach, P., Inoue, K. and Srinivasan, A., ILP turns 20—Biography and future challenges, *Machine Learning*, 86(1):3–23 (2011)
33. Otero, R.P., Induction of the indirect effects of actions by monotonic methods, in: *Proceedings of ILP '05*, LNAI 3625, pp.279–294, Springer (2005)
34. Pal, R., Ivanov, I., Datta, A., Bittner, M.L. and Dougherty, E.R., Generating Boolean networks with a prescribed attractor structure, *Bioinformatics*, 21(21):4021–4025 (2005)
35. Plotkin, G.D., A note on inductive generalization, in: Meltzer, B., Michie, D. (eds.), *Machine Intelligence*, Vol. 5, pp.153–163, Edinburgh University Press (1970)
36. Remy, E. and Ruet, P., From elementary signed circuits to the dynamics of Boolean regulatory networks, *Bioinformatics*, 24:220–226 (2008)
37. Robinson, R.A., A machine-oriented logic based on the resolution principle, *J. ACM*, 12:23–41 (1965)
38. Rodrigues, C., Gérard, P., Rouveirol, C. and Soldano, H., Active learning of relational action models, in: *Proceedings of ILP '11*, LNAI 7207, pp.302–316, Springer (2012)
39. Sakama, C., Nonmonotonic inductive logic programming, in: *Proceedings of LPNMR '01*, LNAI 2173, pp.62–80, Springer (2001)
40. Sakama, C., Induction from answer sets in nonmonotonic logic programs, *ACM Transactions on Computational Logic*, 6(2):203–231 (2005)
41. Sakama, C. and Inoue, K., Abduction, unpredictability and Garden of Eden, *Logic Journal of the IGPL*, to appear (2013)
42. Shmulevich, I., Dougherty, E.R., Kim, S. and Zhang, W., Probabilistic Boolean networks: A rule-based uncertainty model for gene regulatory networks, *Bioinformatics*, 18(2):261–274 (2002)
43. Srinivasan, A. and Bain, M., Knowledge-guided identification of Petri net models of large biological systems, in: *Proceedings of ILP '11*, LNAI 7207, pp.317–331, Springer (2012)
44. Tamaddoni-Nezhad, A., Chaleil, R., Kakas, A. and Muggleton, S. Application of abductive ILP to learning metabolic network inhibition from temporal data, *Machine Learning*, 65:209–230 (2006)
45. Tran, N. and Baral, C., Hypothesizing about signaling networks, *Journal of Applied Logic*, 7(3):253–274 (2009)
46. van Emden, M.H. and Kowalski, R.A., The semantics of predicate logic as a programming language, *Journal of the ACM*, 23(4):733–742 (1976)
47. Wolfram, S., *Cellular Automata And Complexity: Collected Papers*, Westview Press (1994)