Learning from Interpretation Transition

Katsumi Inoue[1] and Chiaki Sakama[2]

[1] National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
[2] Department of Computer and Communication Sciences, Wakayama University
Sakaedani, Wakayama 640-8510, Japan

1 Introduction

There is a growing interest in learning dynamics of systems in the field of inductive logic programming (ILP) [8]. In the view that a logic program is a state transition system [5, 6], given an Herbrand interpretation representing a current state of the world, 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*) [12, 2]. 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. A closer setting can be found in *learning from interpretations* (LFI) [3], in which positive examples are given as Herbrand models of a target program. In fact, we can show that LFI can be constructed from LFIT. Learning action theories [9] can also be related to LFIT, but assumes a sequence of actions in a narrative. 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. Learning NLPs has been considered in ILP, e.g., [11], but most approaches do not take the LFI setting. Moreover, from the semantical viewpoint, our framework can learn NLPs under the *supported model semantics* [2] rather than the stable model semantics [4].

LFIT can be applied to learning rules of dynamic systems such as cellular automata and *Boolean networks* [7], which have been used as a mathematical model of genetic networks and complex adaptive systems. It has been observed in [5] 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. 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 $J = T_P(I)$ holds 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 [1], but no ILP technique has been applied to the 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 another 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.

2 T_P Operator and Attractors

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 positive and negative literals in the body of R of the form (1) 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.

Let $ground(P)$ be the set of ground instances of all rules in a logic program P. 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 [2]. It is known that every stable model [4] is a supported model, 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.

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

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

T_P is monotone when P is definite [12], while it is generally nonmonotone when P is an NLP [2]. 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 $T_P(I) = I$. The *orbit* of I with respect to the T_P operator is the sequence (often called *trajectory*) $\langle {T_P}^k(I) \rangle_{k \in \omega}$, where ${T_P}^0(I) = I$ and ${T_P}^{k+1}(I) = T_P({T_P}^k(I))$ for $k = 0, 1, \ldots$.

A *supported class* of an NLP P [6] is a non-empty set $\mathcal{S}$ of Herbrand interpretations satisfying:

$$\mathcal{S} = \{T_P(I) \mid I \in \mathcal{S}\}. \tag{3}$$

$p \leftarrow q$,
$q \leftarrow p \wedge r$,
$r \leftarrow \neg p$.

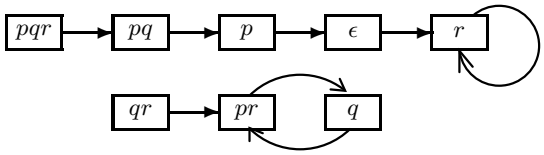

Fig. 1. NLP P_1 (left) and its state transition diagram (right)

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 graph induced by T_P such that $\{I_1, I_2, \ldots, I_k\} = \mathcal{S}$ [6]. A strict supported class thus represents an *attractor* of some orbits, which is either a fixed point (called a *point attractor*) or a periodic oscillation (called a *cycle attractor*).[3] An interpretation that reaches an attractor $\mathcal{S}$ is said to belong to the *basin of attraction* of $\mathcal{S}$.

Example 2.1 Consider the NLP P_1 in Fig. 1 (left). Notice that this logic program 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. Hence, behavior of a corresponding Boolean network is not obvious.

Starting from the interpretation qr,[4] the orbit becomes qr, pr, q, pr, ..., and $pr \rightarrow q \rightarrow pr$ is a cycle attractor (Fig. 1, right below). P_1 has another, point attractor $r \rightarrow r$ (Fig. 1, right above) whose basin of attraction is $\{pqr, pq, p, \epsilon, r\}$.

3 Learning from 1-Step Transitions

Now we consider LF1T as LFIT. 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[5]
Output: An NLP P such that $J = T_P(I)$ holds for any $(I, J) \in E$.

Here we show a bottom-up method to construct an NLP for LF1T.

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 $b^-(R_1)\theta \subseteq b^-(R_2)$. A rule R is the *least (general) generalization* (lg) [10] of R_1 and R_2, written as $R =$

[3] The term "attractor" has been used in dynamical systems as well as in Boolean networks [7]. Based on the equivalence results between Boolean networks and NLPs in [5, 6], we here adopt the notion into state transition induced by the T_P operator.

[4] 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 $\emptyset$.

[5] A negative example (I, J) can 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.

$lg(R_1, R_2)$, 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 [10], 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 [10]). Then, $lg(R_1, R_2)$ is written as in [11]:

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

LF1T(E: pairs of Herbrand interpretations, P: NLP, P_{old}: 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 $R_A^I := A \leftarrow \bigwedge_{B_i \in I} B_i \wedge \bigwedge_{C_j \in \mathcal{B} \setminus I} \neg C_j$;
4. $P :=$ **AddRule**(R_A^I, P); Return to 1.

AddRule(R: rule, P: NLP)
1. If R is subsumed by some rule in P, then return P;
2. Remove from P all rules subsumed by R; Add those removed rules to P_{old};[6]
3. Find a rule $R' \in P \cup P_{old}$ such that (a) $h(R') = h(R)$ and (b) the bodies of R and R' differ in the sign of only one literal. If there is no such a rule in P, then return $P \cup \{R\}$;
4. Return **AddRule**$(lg(R, R'), P \setminus \{R'\})$.

Given the examples E, the algorithm LF1T is initially called by **LF1T**$(E, \emptyset, \emptyset)$. LF1T constructs the most specific rule R_A^I for each positive literal A appearing in $J = T_P(I)$ for each $(I, J) \in E$. The rule R_A^I is then generalized whenever another transition from E makes A true, which is computed by taking the least generalization of the new and some current rules. Taking least generalization $lg(R, R')$ in AddRule is performed in a minimally sufficient way by finding a rule R' such that R' has one and only one complementary literal (conditions (b)).[7] In the propositional case, $lg(R, R')$ is equivalent to the resolvent of R and R' under the condition (b). If these conditions are not satisfied, the least generalization is not added and the new rule R is simply added as an alternative definition for A. By this way, we can guarantee the completeness of the algorithm.

Theorem 3.1 (Completeness of LF1T) *LF1T is complete for any E, that is, it outputs an NLP P such that $J = T_P(I)$ holds for any $(I, J) \in E$.*

Example 3.1 Consider the state transition in Fig. 1 (right). By giving the state transitions step by step, the NLP $P_1 = \{(11), (14), (19)\}$ is obtained in Table 1.

Learning from interpretations (LFI) [3] 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 lost in this conversion. In fact, a usual setting for LFI learns a clausal theory instead of an NLP.

[6] P_{old} globally stores subsumed rules and increases monotonically.

[7] If the condition (b) is dispensed with, simpler rules can be generated more quickly at the expense of loss of completeness. To assure the completeness in this case, a lg can only be added if it is tested the compatibility with the processed examples.

Table 1. Execution of LF1T in inferring P_1 of Example 2.1

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		
		$lg(4,1)$	$p \leftarrow \neg p \wedge q$	5	2,3,5	+ 1,4
		R_r^{q}	$r \leftarrow \neg p \wedge q \wedge \neg r$	6		
		$lg(6,2)$	$r \leftarrow \neg p \wedge q$	7	3,5,7	+ 2,6
4	$pqr \to pq$	R_p^{pqr}	$p \leftarrow p \wedge q \wedge r$	8		
		$lg(8,1)$	$p \leftarrow q \wedge r$	9	3,5,7,9	+ 8
		R_q^{pqr}	$q \leftarrow p \wedge q \wedge r$	10		
		$lg(10,3)$	$q \leftarrow p \wedge r$	11	5,7,9,11	+ 3,10
5	$pq \to p$	R_p^{pq}	$p \leftarrow p \wedge q \wedge \neg r$	12		
		$lg(12,4)$	$p \leftarrow q \wedge \neg r$	13	5,7,9,11,13	+ 12
		$lg(13,9)$	$p \leftarrow q$	14	7,11,14	+ 5,9,13
6	$p \to \epsilon$				7,11,14	
7	$\epsilon \to r$	R_r^{ϵ}	$r \leftarrow \neg p \wedge \neg q \wedge \neg r$	15		
		$lg(15,6)$	$r \leftarrow \neg p \wedge \neg r$	16	7,11,14,16	+ 15
8	$r \to r$	R_r^{r}	$r \leftarrow \neg p \wedge \neg q \wedge r$	17		
		$lg(17,15)$	$r \leftarrow \neg p \wedge \neg q$	18	7,11,14,16,18	+ 17
		$lg(18,7)$	$r \leftarrow \neg p$	19	11,14,19	+ 7,16,18

4 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 [1]. 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.

Learning from Basins of Attraction (LFBA)

Input: $\mathcal{E} \subseteq 2^{2^{\mathcal{B}}}$: (positive) examples/observations

Output: An NLP P such that any $I \in \mathcal{I}$ belongs to the basin of attraction of an attractor of P that is contained in $\mathcal{I}$ for every $\mathcal{I} \in \mathcal{E}$.

In LFBA, an example $\mathcal{I}$ is given as a part of the basin of attraction of some attractor of the target NLP P. We here assume 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 $P := \emptyset$; $P_{old} := \emptyset$;
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. $P := \mathbf{LF1T}(E, P, P_{old})$; Return to 2.

Theorem 4.1 (Completeness of LFBA) *LFBA is complete for any* $\mathcal{E}$.

5 Concluding Remark

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 dynamic systems in terms of NLP learning from synchronous state transitions. More complex schemes such as asynchronous and probabilistic updates do not obey transition by the T_P operator. Learning such dynamic networks is important extension of the work. Incorporation of biases and background knowledge is also future work.

References

1. Akutsu, T., Kuhara, S., Maruyama, O., Miyano, S.: Identification of genetic networks by strategic gene disruptions and gene overexpressions under a Boolean model. *Theoretical Computer Science*, 298:235–251 (2003)
2. Apt, K.R., Blair, H.A., 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)
3. De Raedt, L.: Logical settings for concept-learning. *Artificial Intelligence*, 95:187–201 (1997)
4. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: *Proceedings of ICLP '88*, pp. 1070–1080, MIT Press (1988)
5. Inoue, K.: Logic programming for Boolean networks. In: *Proceedings of IJCAI-11*, pp. 924–930, AAAI Press (2011)
6. Inoue, K., 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, vol. 7265, pp. 345–362, Springer (2012)
7. Kauffman, S.A.: *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press (1993)
8. Muggleton, S.H., De Raedt, L., Poole, D., Bratko, I., Flach, P., Inoue, K., Srinivasan, A.: ILP turns 20—Biography and future challenges. *Machine Learning*, 86(1):3–23 (2011)
9. Otero, R.P.: Induction of the indirect effects of actions by monotonic methods. In: *Proceedings of ILP '05*. LNAI, vol. 3625, pp. 279–294, Springer (2005)
10. 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)
11. Sakama, C.: Nonmonotonic inductive logic programming. In: *Proceedings of LPNMR '01*. LNAI, vol. 2173, pp. 62–80, Springer (2001)
12. van Emden, M.H., Kowalski, R.A.: The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742 (1976)