Learning Delayed Influence of Dynamical Systems From Interpretation Transition

Tony Ribeiro[1], Morgan Magnin[2,3], and Katsumi Inoue[1,2]

[1] The Graduate University for Advanced Studies (Sokendai),
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
tony_ribeiro@nii.ac.jp,
[2] National Institute of Informatics,
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan,
{magnin,inoue}@nii.ac.jp
[3] LUNAM Université, École Centrale de Nantes, IRCCyN UMR CNRS 6597
1 rue de la Noë - B.P. 92101 - 44321 Nantes Cedex 3, France.

Abstract. In some biological and physical phenomena, effects of actions or events appear at some later time points. Such delayed influence can play a major role in various biological systems of crucial importance. Learning such dynamics is the purpose of our work. We propose an extension of the learning from interpretation transition approach that has been proposed previously. This method considers as input a set of state transitions and builds a normal logic program that realizes the given transition relations. The novelty of this work is that the new algorithm we propose is able to consider k-step transitions, while the previous one dealt only with 1-step transitions. Hence, we are now able to capture delayed influence with an inductive logic programming methodology.

Keywords: dynamical systems, Boolean networks, delay, Markov(k), learning from interpretation transition, Inductive Logic Programming

1 Introduction

In some biological and physical phenomena, effects of actions or events appear at some later time points. For example, delayed influence can play a major role in various biological systems of crucial importance, like the mammalian circadian clock [4] or the DNA damage repair [1]. Social interactions too may depend of the behaviors history of the agents at stake [9]. While Boolean networks have proven to be a simple, yet powerful, framework to model and analyze the dynamics of the above examples, they usually assume that the modification of one node results in an immediate activation (or inhibition) of its targeted nodes [2] for the sake of simplicity But this hypothesis is sometimes too broad and we really need to capture the memory of the system i.e., keep track of the previous steps, to get a more realistic model. Our work aims to give an efficient and valuable approach to learn such dynamics.

The most used framework to model delayed and indirect influences in Boolean networks was designed by A. Silvescu et al. [11]: the authors introduced an

extension of Boolean networks from a Markov(1) to Markov(k) model, where k is the number of time steps during which a variable can influence another variable. This extension is called temporal Boolean networks, abridged as $TBN(n, m, k)$, with n the number of variables and the expression of each variable at time $t+1$ being controlled by a Boolean function of the expression levels of at most m variables at times in $\{t, t-1, \ldots, t-(k-1)\}$. In this paper, we will consider $TBN(n, m, k)$ and discuss a new learning algorithm.

In some previous works, Markov(1) state transition systems are represented with logic programs [5, 7], in which the state of the world is represented by an Herbrand interpretation and the dynamics that rule the environment changes are represented by a logic program P. The rules in P specify the next state of the world as an Herbrand interpretation through the *immediate consequence operator* (also called the T_P *operator*) [12, 3]. With such a background, Inoue *et al.* [6] have recently proposed a framework to learn logic programs from traces of interpretation transitions (LFIT). The learning setting of this framework is as follows. We are given a set of pairs of Herbrand interpretations (I, J) as positive examples such that $J = T_P(I)$, and the goal is to induce a *normal logic program* (NLP) P that realizes the given transition relations.

We extend these researches by designing an algorithm that takes multiple sequence of state transition as input and builds a normal logic program that capture the delayed dynamics of a Markov(k) system. While the previous algorithm dealt only with 1-step transitions (i.e., we assume the state of the system at time t depends only of its state at time $t-1$), we propose here an approach that is able to consider k-step transitions (sequence of at most k state transitions). This means that we are now able to capture delayed influences in the inductive logic programming methodology. Because of the lack of space, this short paper will focus on the methodology. In further works, we plan to learn Boolean networks with delayed influences from real biological case studies.

The paper is organized as follows: Section 2 reviews the logical background of this work, and summarizes the main ideas behind the existing LF1T algorithm in order to make its extension to Markov(k) models (i.e., with delayed influences) in Section 3 be more understandable. Finally, we give the complete algorithm in Section 4 and discuss these results and further works in Section 5.

2 Background

2.1 Logic Programming

In this section we recall some preliminaries about logic programming. 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. The set of ground instances of all rules in a logic program P is denoted as $ground(P)$.

An (*Herbrand*) *interpretation* I is a subset of $\mathcal{B}$. For a logic program P and an Herbrand interpretation I, the *immediate consequence operator* (or T_P *operator*) [3] is the mapping $T_P : 2^{\mathcal{B}} \rightarrow 2^{\mathcal{B}}$:

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

Definition 1 (Subsumption). *Let R_1 and R_2 be two ground rules. If $h(R_1) = h(R_2)$ and $b(R_1) \subseteq b(R_2)$ then R_1 subsumes R_2.*

Example 1. Let R_1 and R_2 be the two following rules: $R_1 = (a \leftarrow b)$, $R_2 = (a \leftarrow a \wedge b)$, R_1 subsumes R_2 because $(b(R_1) = \{b\}) \in (b(R_2) = \{a, b\})$.

Definition 2 (Minimal Specific Specialization). *Let S and G be the body of two rules. S is a* **specialization** *of G if $G \subset S$. S is a* **minimal specific specialization** *of G if S is a specialization of G, and there is no other specialization S' of G such that S is a specialization of S', i.e. $\nexists S', G \subset S' \subset S$.*

Example 2. $C_1 = (a \wedge b \wedge c)$ is a minimal specific specialization of $C_2 = (a \wedge b)$. But C_1 is not a minimal specific specialization of $C_3 = (a)$. Because C_2 is a specialization of C_3 and C_1 is a specialization of C_2.

2.2 Learning from Interpretation Transitions

LF1T [6] is an *any time algorithm* that takes a set of one-step state transitions E as input. From these transitions the algorithm learns a logic program P that represents the dynamics of E. Figure 1 represents a Boolean network with its corresponding state transition diagram. Given this state transition diagram as input, **LF1T** can learn the Boolean network B_1.

Learning from 1-Step Transitions (LF1T)

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

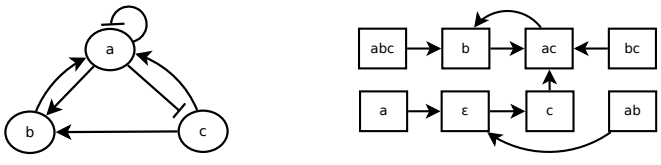

Fig. 1. A Boolean Network B_1(left) and its state transition diagram (right)

3 Learning Markov(k) systems

A Markov(1) system can be represented by a deterministic state transition diagram, like the one of Figure 1. A *Markov(k) system* can be seen as a k-step deterministic system. In other words, the state of the system may depend on its (at most) k previous states. If a Boolean network is Markov(k), it means that k is the maximum number of time steps such that the influence of any component (e.g., a gene) on another component is expressed. Example 3 shows a Markov(2) system with 2 variables a and b.

Definition 3 (Markov(k) system). *A Markov(k) system S is an NLP, where each rule R in S is a rule such that any atom appearing in $h(R) \in \mathcal{B}$, with $\mathcal{B}$ the variables of S, and $b(R) \in B_k$, with $\mathcal{B}_k = \{\bigcup_{i=1}^{k} v_{t-i} | v \in \mathcal{B}\}$. We call $\mathcal{B}$ the 0-step Herbrand Base of S and $\mathcal{B}_k$ the k-step Herbrand Base of S.*

Example 3. Let R_1 and R_2 be two rules, $R_1 := a \leftarrow b_{t-1}, b_{t-2}$, $R_2 := b \leftarrow a_{t-2}, \neg b_{t-2}$. The NLP $S := \{R_1, R_2\}$ is a Markov(2) system, i.e. the state of the system depends on the two previous states. The value of a is true at time step t only if b was true at $t-1$ and $t-2$. The value of b is true at time step t only if a was true at $t-2$ and b was false at $t-2$. The 0-step Herbrand base of S is $\mathcal{B} = \{a, b\}$. The 1-step Herbrand base of S is $\mathcal{B}_1 = \{a_{t-1}, b_{t-1}\}$ and the 2-step Herbrand base of S is $\mathcal{B}_2 = \{a_{t-1}, b_{t-1}, a_{t-2}, b_{t-2}\}$.

Definition 4 (Trace of execution). *A trace of execution T of a Markov(k) system S is a finite sequence of states of S. Let $\mathcal{B}$ be the Herbrand base of S, $T := (S_0, \ldots, S_n), n \geq 1, S_i \in 2^{\mathcal{B}}$. We note $|T|$ the size of the trace, that is the number of elements of the sequence. A sub-trace of size k of a trace of execution T is a sub-sequence of T of size k, where $k \leq |T|$.*

Example 4. Let $T_1 := a \rightarrow b \rightarrow a$ be a trace of execution. T_1 is a trace of size 2 and $a \rightarrow b$ and $b \rightarrow a$ are sub-traces of size 1 of T_1.

Definition 5 (Consistent traces). *Let T_1, T_2 be two traces of execution. Let $ST \in T_1$ be a state transition and let $prev(ST)$ be the sequence of state transitions in T_1 that appears before ST. T_1 and T_2 are consistent if, for all state transitions $(I \rightarrow J) \in T_1$, if there exists $(I \rightarrow J') \in T_2$ such that $J' \neq J$ then $prev(I \rightarrow J) \neq prev(I \rightarrow J')$.*

- $a \rightarrow b \rightarrow b$
- $ab \rightarrow b \rightarrow a$
- $b \rightarrow b \rightarrow a$
- $\epsilon \rightarrow b \rightarrow \epsilon$
- $b \rightarrow \epsilon \rightarrow \epsilon$
- $ab \rightarrow \epsilon \rightarrow \epsilon$
- $a \rightarrow \epsilon \rightarrow b$
- $\epsilon \rightarrow \epsilon \rightarrow \epsilon$

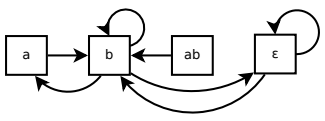

Fig. 2. Traces of execution and state transition diagram of the system of Example 3

As shown in Figure 2, Markov(k) system can appears to be non deterministic when its represented by a state transition diagram. That is because such state transition diagram only represent 1-step transitions. In this example, the transition from the state b is not Markov(1): the next state can be either a or b. But it can be Markov(2), because all the traces of size 2 of Figure 2 are consistent.

Definition 6 (k-step interpretation transitions). *Let S be a system and $\mathcal{B}$ be the Herbrand base of S. Let $\mathcal{B}_k$ be the k-step Herbrand base of S.A k-step interpretation transition is a pair of Herbrand interpretation (I, J) where $I \subseteq \mathcal{B}_k$ and $J \subseteq \mathcal{B}$.*

Example 5. The trace $ab \rightarrow b \rightarrow a$ can be interpreted in three ways as follows:
- $(a_{t-2}b_{t-2}b_{t-1}, a)$: the 2-step interpretation that corresponds to the full trace.
- $(a_{t-1}b_{t-1}, b)$: the 1-step interpretation corresponding to the sub-trace $ab \rightarrow b$.
- (b_{t-1}, a): the 1-step interpretation that corresponds to the sub-trace $b \rightarrow a$.

Definition 7 (Consistency). *Let R be a rule and (I, J) be a k-step interpretation transition. R is consistent with (I, J) if $I \not\models b(R)$ or if $I \models b(R)$ and $h(R) \in J$. Let T be a sequence of state transitions, R is consistent with T if it is consistent with every k-step interpretation transition of T. Let O be a set of sequences of state transitions, R is consistent with O if R is consistent with all $T' \in O$.*

Our goal is to learn the dynamics of a Markov(k) system from its traces of execution. The main idea is to extract n-step interpretation transitions, $1 \leq n \leq k$, from the traces of executions of the system. Transforming the trace into pair of interpretations allows us to use least specialization [10] to iteratively learn the dynamics of the system. In short, the idea is to generates hypotheses by *specialization* from the most general clauses until no negative transition is covered. We analyze each interpretation transition one by one and revise the learned rules when they subsume a negative example. Here, least specialization is used, which consists in replacing a rule by all its minimal specific specializations that do not subsume the negative example. The difference is that we have now to deal with n-step interpretation transitions where only 1-step interpretation transitions were considered previously.

4 Algorithm

In this section we propose **LFkT** an extension of the **LF1T** algorithm to learn a Markov(k) system. **LFkT** takes a set of traces of executions O as input, each trace is a sequence of state transitions. If all traces are consistent, the algorithm outputs an NLP P that realizes all transitions of O. The learned influences can be at most k-step relations, where k is the size of the longest trace of O. Algorithm 1 shows the pseudo-code of **LFkT** and Algorithm 2 shows how input interpretation is done in practice (pseudo code is given in appendix).

Step 1: The algorithm starts with k NLPs $P_0^{\mathcal{B}}$. The idea is to learn rules independently for each possible k-step relation: 1-step rules, 2-step rules, ...,

k-step rules. The rules learned from 1-step interpretations will go into the 1-step program, the rules learned from 2-step interpretations will go into the 2-step program and so on. These different programs are merged at the end to constitute a NLP that realizes all consistent traces of O.

Step 2: In order to use least specialization, we need to convert the input traces of execution into interpretation transitions. This conversion is done by the function **interprete**, whose pseudo code is given in Algorithm 2. It extracts all k-step interpretations from each trace $T \in O$. It can be done by extracting and converting all sub-traces of T into corresponding interpretations. The function outputs them as a vector of set of interpretation transitions E, where each set E_i corresponding to interpretation of sub-trace of size i.

Step 3: The algorithm iteratively learns from each set of pair of interpretations $E_i \in E$. Now it only needs to apply the **LF1T** method of [10] on each set E_i by analyzing each pair of interpretations $(I, J) \in E_i$. For each variable A that **does not appear** in J, it infers an **anti-rule** $R_A^I := A \leftarrow \bigwedge_{B_i \in I} B_i \wedge \bigwedge_{C_j \in (\mathcal{B}_i \setminus I)} \neg C_j$, where $\mathcal{B}_i$ is the i-step Herbrand base of E_i, i.e. all atoms that can appear in a rule of E_i. Then, least specialization is used to make the corresponding NLP P_i' consistent with R_A^I. Algorithm 2 of [10] shows the pseudo code of this operation (also given as appendix).

Step 4: After analyzing all interpretation transitions, the programs that have been learned are merged into a unique NLP. This operation ensures that the rule outputted are consistent with all observations. If a rule is not consistent with an observation, it has to be deleted. It can be checked by comparing each rule with other NLPs. If a n-step rule R is more general than a n'-step rules R', $n' < n$, then R is not consistent with the observations from which R' has been learned. To avoid this case, we just need to remove n-step rules that have no v_n variable. Finally, **LFkT** output an NLP that realizes all consistent traces of execution of O. If O is a set of traces of execution of a Boolean network, the NLP outputted by **LFkT** represents the Boolean functions of each variables. For each variable, it correspond to the conditions over the k previous step to make it active at $t+1$.

Theorem 1 (Correctness of LFkT). *Let O be a set of trace of execution of a Markov(k) system S. Using O as input,* **LFkT** *output an NLP that realizes all consistent traces of O. Proof is given in appendix.*

5 Conclusion and Future Work

This short paper gives an overview of our approach to learn normal logic programs from interpretation transitions on k-steps. This can be directly applied to the learning of Boolean networks with delayed influence, which is crucial to understand the memory effect involved in some interactions between biological components. Further works aim at adapting the approach developed in the paper to the kind of data as produced by biologists [8]. This requires to connect through various databases in order to extract real time series data, and subsequently explore and use them to learn genetic regulatory networks. We also consider extending the methodology to asynchronous semantics, which can help to capture more realistic behaviors.

References

1. Abou-Jaoudé, W., Ouattara, D.A., Kaufman, M.: From structure to dynamics: frequency tuning in the p53–mdm2 network: I. logical approach. Journal of theoretical biology 258(4), 561–577 (2009)
2. Akutsu, T., Kuhara, S., Maruyama, O., Miyano, S.: Identification of genetic networks by strategic gene disruptions and gene overexpressions under a boolean model. Theoretical Computer Science 298(1), 235–251 (2003)
3. Apt, K.R., Blair, H.A., Walker, A.: Towards a theory of declarative knowledge. Foundations of deductive databases and logic programming p. 89 (1988)
4. Comet, J.P., Bernot, G., Das, A., Diener, F., Massot, C., Cessieux, A.: Simplified models for the mammalian circadian clock. Procedia Computer Science 11, 127–138 (2012)
5. Inoue, K.: Logic programming for boolean networks. In: IJCAI Proceedings-International Joint Conference on Artificial Intelligence. vol. 22, p. 924 (2011)
6. Inoue, K., Ribeiro, T., Sakama, C.: Learning from interpretation transition. Machine Learning 94(1), 51–79 (2014)
7. Inoue, K., Sakama, C.: Oscillating behavior of logic programs. In: Correct Reasoning, pp. 345–362. Springer (2012)
8. Li, X., Rao, S., Jiang, W., Li, C., Xiao, Y., Guo, Z., Zhang, Q., Wang, L., Du, L., Li, J., et al.: Discovery of time-delayed gene regulatory networks based on temporal gene expression profiling. BMC bioinformatics 7(1), 26 (2006)
9. Paczuski, M., Bassler, K.E., Corral, A.: Self-organized networks of competing boolean agents. Phys. Rev. Lett. 84, 3185–3188 (Apr 2000), http://link.aps.org/doi/10.1103/PhysRevLett.84.3185
10. Ribeiro, T., Inoue, K.: Learning prime implicant conditions from interpretation transition. In: The 24th International Conference on Inductive Logic Programming (2014), to appear (long paper) (http://tony.research.free.fr/paper/ILP2014)
11. Silvescu, A., Honavar, V.: Temporal boolean network models of genetic networks and their inference from gene expression time series. Complex Systems 13(1), 61–78 (2001)
12. Van Emden, M.H., Kowalski, R.A.: The semantics of predicate logic as a programming language. Journal of the ACM (JACM) 23(4), 733–742 (1976)

A Appendix

A.1 Proof of Theorem 1 (Correctness of LFkT)

Let O be a set of trace of execution of a Markov(k) system S. Using O as input, **LFkT** output a NLP that realizes all consistent traces of O.

Proof. Let V be the vector of interpretation transition extracted from O by **LFkT** (Algorithm 2). According to Theorem 4 of [10], initializing **LF1T** with $P_0^{\mathcal{B}}$, by using least specialization iteratively on a set of interpretation transitions E, we obtain a NLP P that realizes E. Since **LFkT** use **LF1T** on each element of V, **LFkT** learn a vector of NLP P' such that each NLP $p'_n \in P'$ realizes the corresponding set of interpretation transitions $v_n \in V$.

Let $p'_n \in P'$ be the NLP learn from $v_n \in V$, $n \geq 1$. p'_n is obtained by least specialization of $P_0^{\mathcal{B}}$ with all anti-rule of v_n (non consistent rule). According to Theorem 3 of [10], p'_n does not subsume any anti-rule of v_n. Then, p'_n realizes all deterministic transition of v_n, that is $\forall (I, J) \in v_n, \nexists (I, J'), J \neq J'$ and p'_n only contains prime rules.

Since v_n contains interpretation transition that represent all sub-traces of size n of O, p'_n realizes all consistent sub-trace of size n of O. Let P_{n-1} be a NLP that realizes all consistent sub-traces of size at most $n-1$ of O. p'_n can contains a rule R such that $(\mathcal{B}_n \setminus \mathcal{B}_{n-1}) \cap b(R) = \emptyset$ (no literal of R refers to the $t-n$ state of the variables). In this case R realizes a sub-trace of size n and also some sub-traces of size at most $n-1$. If these sub-traces of size $n-1$ are consistent, then they are necessary realized by P_{n-1}. $P_{n-1} \cup \{R\}$ does not realize more consistent sub-trace of size at most $n-1$ than P_{n-1}. Let S_R be the set of rules of p'_n of the form R, then $(p'_n \setminus S_R)$ only realizes all sub-traces of size n of O. Then the NLP $P_n = P_{n-1} \cup (p'_n \setminus S_R)$ only realizes all consistent sub-trace of size at most $n-1$ of O and all sub-traces of size n of O, that is P_n realizes all consistent sub-traces of size at most n of O.

Let $p'_1 \in P'$ be the NLP learn from $v_1 \in V$, and let $P = p'_1$. Let R' be all rule of the NLP p'_n such that $(\mathcal{B}_n \setminus \mathcal{B}_{n-1}) \cap b(R') \neq \emptyset$. Iteratively adding rules R' into P, starting by the NLP p_2 until p_k, we obtain a NLP that realizes all consistent sub-traces of size at most k of O. So that, using O as input, **LFkT** output a NLP that realizes all consistent traces of O. □

A.2 Running example

Table A.2 shows the execution of **LFkT** on traces of figure 2 where $(a_2b_2b_1, a)$ represents the interpretation of the trace $ab \rightarrow b \rightarrow a$. Introduction of literal by least specialization is represented in bold and rules that are subsumed after specialization are stroked. For the sake of space use, here a_1 and a_2 respectively correspond to a_{t-1} and a_{t-2}.

Initialization	
2-step NLP	1-step NLP
$a.$ $b.$	$a.$ $b.$

$a \rightarrow b \rightarrow b$	
(a_2b_1, b)	(a_1, b)
$a \leftarrow \neg\mathbf{a_2}.$ $a \leftarrow \mathbf{b_2}.$ $a \leftarrow \mathbf{a_1}.$ $a \leftarrow \neg\mathbf{b_1}.$ $b.$	$a \leftarrow \neg\mathbf{a_1}.$ $a \leftarrow \mathbf{b_1}.$ $b.$ (b_1, b) $a \leftarrow \neg a_1, \neg\mathbf{b_1}.$ $a \leftarrow \mathbf{a_1}, b_1.$ $b.$

$ab \rightarrow b \rightarrow a$	
$(a_2b_2b_1, a)$	(a_1b_1, b)
$a \leftarrow \neg a_2.$ $a \leftarrow b_2.$ $a \leftarrow a_1.$ $a \leftarrow \neg b_1.$ $b \leftarrow \neg a_2.$ $b \leftarrow \neg b_2.$ $b \leftarrow a_1.$ $b \leftarrow \neg b_1.$	$a \leftarrow \neg a_1, \neg b_1.$ ~~$a \leftarrow a_1, b_1.$~~ $b.$ (b_1, a) $a \leftarrow \neg a_1, \neg b_1.$ $b \leftarrow \mathbf{a_1}.$ $b \leftarrow \neg\mathbf{b_1}.$

$b \rightarrow b \rightarrow a$	
(b_2b_1, a)	(b_1, b)
$a \leftarrow \neg a_2.$ $a \leftarrow b_2.$ $a \leftarrow \neg a_1.$ $a \leftarrow \neg b_1.$ ~~$b \leftarrow \neg a_2, \neg\mathbf{b_2}.$~~ ~~$b \leftarrow \neg a_2, \mathbf{a_1}.$~~ ~~$b \leftarrow \neg a_2, \neg\mathbf{b_1}.$~~ $b \leftarrow \neg b_2.$ $b \leftarrow a_1.$ $b \leftarrow \neg b_1.$	$a \leftarrow \neg a_1, \neg b_1.$ $b \leftarrow a_1.$ $b \leftarrow \neg b_1.$ (b_1, a) $a \leftarrow \neg a_1, \neg b_1.$ $b \leftarrow a_1.$ $b \leftarrow \neg b_1.$

$\epsilon \rightarrow \epsilon \rightarrow \epsilon$	
(ϵ, ϵ)	(ϵ, ϵ)
~~$a \leftarrow \neg a_2, \mathbf{b_2}.$~~ ~~$a \leftarrow \neg a_2, \mathbf{a_1}.$~~ $a \leftarrow \neg a_2, \mathbf{b_1}.$ $a \leftarrow b_2.$ $a \leftarrow a_1.$ $a \leftarrow \mathbf{a_2}, \neg b_1.$ ~~$a \leftarrow \mathbf{b_2}, \neg b_1.$~~ ~~$a \leftarrow \mathbf{a_1}, \neg b_1.$~~ $b \leftarrow \mathbf{a_2}, \neg b_2.$ ~~$b \leftarrow \neg b_2, \mathbf{a_1}.$~~ $b \leftarrow \neg b_2, \mathbf{b_1}.$ $b \leftarrow a_1.$ $b \leftarrow \mathbf{a_2}, \neg b_1.$ $b \leftarrow \mathbf{b_2}, \neg b_1.$ ~~$b \leftarrow \mathbf{a_1}, \neg b_1.$~~	~~$a \leftarrow \neg a_1, \neg b_1.$~~ $b \leftarrow a_1.$ ~~$b \leftarrow \mathbf{a_1}, \neg b_1.$~~ (ϵ, ϵ) $b \leftarrow a_1.$

$b \rightarrow \epsilon \rightarrow \epsilon$	
(b_2, ϵ)	(b_1, ϵ)
$a \leftarrow \neg a_2, b_1.$ $a \leftarrow \mathbf{a_2}, b_2.$ ~~$a \leftarrow b_2, \mathbf{a_1}.$~~ $a \leftarrow b_2, \mathbf{b_1}.$ $a \leftarrow a_1.$ $a \leftarrow \mathbf{a_2}, \neg b_1.$ $b \leftarrow \mathbf{a_2}, \neg b_2.$ $b \leftarrow b_2, b_1.$ $b \leftarrow a_1.$ $b \leftarrow a_2, \neg b_1.$ ~~$b \leftarrow \mathbf{a_2}, b_2, \neg b_1.$~~ ~~$b \leftarrow b_2, \neg b_1, \mathbf{a_1}.$~~	$b \leftarrow a_1.$ (ϵ, ϵ) $b \leftarrow a_1.$

$ab \rightarrow \epsilon \rightarrow \epsilon$	
(a_2b_2, ϵ)	(a_1b_1, ϵ)
$a \leftarrow \neg a_2, b_1.$ ~~$a \leftarrow a_2, b_2, \mathbf{a_1}.$~~ ~~$a \leftarrow a_2, b_2, \mathbf{b_1}.$~~ $a \leftarrow b_2, b_1.$ $a \leftarrow a_1.$ $a \leftarrow a_2, \neg\mathbf{b_2}, \neg b_1.$ ~~$a \leftarrow a_2, \mathbf{a_1}, \neg b_1.$~~ $b \leftarrow \mathbf{a_2}, \neg b_2.$ $b \leftarrow \neg b_2, b_1.$ $b \leftarrow a_1.$ ~~$b \leftarrow a_2, \neg\mathbf{b_2}, \neg b_1.$~~ ~~$b \leftarrow a_2, \mathbf{a_1}, \neg b_1.$~~	$b \leftarrow a_1, \neg\mathbf{b_1}.$ (ϵ, ϵ) $b \leftarrow a_1, \neg b_1.$

$a \rightarrow \epsilon \rightarrow b$	
(a_2, ϵ)	(a_1, ϵ)
$a \leftarrow \neg a_2, b_1.$ $a \leftarrow b_2, b_1.$ $a \leftarrow a_1.$ ~~$a \leftarrow a_2, \neg b_2, \mathbf{a_1} \neg b_1.$~~ $b \leftarrow a_2, \neg b_2.$ $b \leftarrow \neg b_2, b_1.$ $b \leftarrow a_1.$	~~$b \leftarrow a_1, \neg b_1.$~~ (ϵ, b) $\emptyset$

$\epsilon \rightarrow b \rightarrow \epsilon$	
(b_1, ϵ)	(ϵ, b)
~~$a \leftarrow \neg a_2, \mathbf{b_2} b_1.$~~ ~~$a \leftarrow \neg a_2, \mathbf{a_1} b_1.$~~ $a \leftarrow b_2, b_1.$ $a \leftarrow a_1.$ $b \leftarrow a_2, \neg b_2.$ ~~$b \leftarrow \mathbf{a_2}, \neg b_2, b_1.$~~ ~~$b \leftarrow \neg b_2, \mathbf{a_1}, b_1.$~~ $b \leftarrow a_1.$	$\emptyset$ (b_1, ϵ) $\emptyset$

Merging of the programs	OUTPUT
$a \leftarrow b_2, b_1.$ ~~$a \leftarrow a_1.$~~ $b \leftarrow a_2, \neg b_2.$ ~~$b \leftarrow a_1.$~~	$a \leftarrow b_2, b_1.$ $b \leftarrow a_2, \neg b_2.$

A.3 Pseudo Code

Algorithm 1 LFkT(O) : Learn the most general rules that explain E

INPUT: O a set of sequences of state transition (traces of executions)
OUTPUT: The complete minimal NLP that realizes the transitions of O.

P' a vector of set of rule
E a vector of set of pair of interpretations (I, J)
$max_k :=$ the size of the longest trace of O
// 1) Initialize P' with max_k NLP that contain the most general rules
for each k from 1 to max_k **do**
    $P'_k := \emptyset$
    for each $A \in \mathcal{B}$ **do**
        $P'_k := P_k \cup \{A.\}$
    $P' := P' \cup P'_k$
// 2) Extract interpretation from trace of executions
$E :=$ **interprete**(O)

// 3) Specify P' by the interpretation of the trace of executions
for each k from 1 to max_k **do**
    $E_k :=$ the k^{th} set of interpretation of E
    while $E_k \neq \emptyset$ **do**
        Pick $(I, J) \in E_k$; $E_k := E_k \setminus \{(I, J)\}$
        for each $A \in \mathcal{B}$ **do**
            if $A \notin J$ **then**
                $R^I_A := A \leftarrow \bigwedge_{B_i \in I} B_i \wedge \bigwedge_{C_j \in (\mathcal{B} \setminus I)} \neg C_j$
                $P'_k :=$ the k^{th} set of rule of P'
                $P'_k :=$ **Specialize**(P'_k, R^I_A)
    end while

// 4) Merge the programs into a unique NLP
$P := \emptyset$
for each k from 1 to max_k **do**
    $P'_k :=$ the k^{th} set of rule of P'
    Remove from P'_k all rules that does not contain any literal of the form v_{t-k}
    $P := P \cup P'_k$
return P

Algorithm 2 interprete(O) : Extract interpretations transition from traces

INPUT: O a set of sequences of state transition (traces of executions)
OUTPUT: E a vector of set of pair of interpretations (I, J)

$E := \emptyset$
// Extract interpretations
for each sequence $T \in O$ **do**
  for each k from $|T|$ to 1 **do**
    for each sub-trace T' of size k in T **do**
      $s_k :=$ the k^{th} state of T
      $I := \emptyset$
      for each state $s_{k'}$ before s_k in T' **do**
        $t := k - k'$
        for each atom $a \in s_{k'}$ **do**
          $I := I \cup \{a_t\}$
        $E_k :=$ the k^{th} set of interpretation of E
        $E_k := E_k \cup \{(I, s_k)\}$
return E

Algorithm 3 specialize(P,R) : specify the NLP P to not subsume the rule R

INPUT: a NLP P and a rule R
OUTPUT: the minimal specific specialization of the rule of P by R.

$conflicts$: a set of rules
$conflicts := \emptyset$
// Search rules that need to be specialized
for each rule $R_P \in P$ **do**
  if R_P is conflicting with R **then**
    $conflicts := conflicts \cup R_P$
    $P := P \setminus R_P$
// Revise the rules by least specialization
for each rule $R_c \in conflicts$ **do**
  for each literal $l \in b(R)$ **do**
    if $l \notin b(R_c)$ and $\bar{l} \notin b(R_c)$ **then**
      $R'_c := (h(R_c) \leftarrow (b(R_c) \cup \bar{l}))$
      if P does not subsume R'_c **then**
        $P := P \setminus$ all rules subsumed by R'_c
        $P := P \cup R'_c$
return P