Learning Stochastic Logical Automaton

Hiroaki Watanabe and Stephen Muggleton

Imperial College London, 180 Queen's Gate, London SW7 2AZ, UK
Email: {hw3,shm}@doc.ic.ac.uk
Telephone: +44 (0)20 7594 8287
Fax: +44 (0)20 7581 8024

Abstract. This paper is concerned with algorithms for the logical generalisation of probabilistic temporal models from examples. The algorithms combine logic and probabilistic models through inductive generalisation. The inductive generalisation algorithms consist of three parts. The first part describes the graphical generalisation of state transition models. State transition models are generalised by applying state mergers. The second part involves symbolic generalisation of logic programs that are embedded in each state. Plotkin’s RLLG is used for symbolic generalisation of logic programs. The third part covers learning of parameters using statistics derived from the input sequences. The state transitions are unobservable in our settings. The probability distributions over the state transitions and actions are estimated using the EM algorithm. These algorithms are implemented in a Machine Learning System “Cellist” for showing the benefits of combining logic and probabilistic models in uncertain domains. As an application of the system, we learn chemical reaction rules from the stochastic software simulator of biochemical reactions.

1 Introduction

Logical Induction from temporal observations is a challenging problem since the observations could contain uncertainties in many cases. The temporal observations involve both dynamic and static information in their nature, however, the distinction between the dynamic and static aspects might be difficult under uncertainties. One way to tackle the difficulties is to derive statistics from the observations. If we express the statistical knowledge explicitly, we would need to combine stochastic and logical knowledge representations.

Logic-based AI has been studying logical representations of dynamic aspects of temporal data since McCarthy and Hayes propose Situation Calculus[1] in first-order logic where dynamic changing world is expressed in time-sliced declarative representations. In the ILP literatures, a few studies have been reported from the time-sliced representation point of view [2,3]. Automata-based representations have also been studied in computer science. One of the merits of the automata-based representations under uncertainties would be the applicability of the well-studied statistical learning algorithms such as EM algorithm for HMMs.

In this paper, from the automata-based representation point of view we propose a logical and graphical knowledge representation as well as its induction algorithms. More precisely, we introduce a logical extension of stochastic non-deterministic finite automata. Dynamic aspects of the model are generalised by the state merging technique whereas static knowledge are generalised by Plotkin’s RLLG [4] with ground level background knowledge.

The paper is organised as follows. Section2 introduces our logical automata. Section3 explains the overview of the inductive algorithms of our model. Section4 contains our initial attempt to learn chemical reaction rules from the biochemical simulator StochSim[5]. We discuss some related works in Section5. Discussions and future work conclude the paper in Section6.

2 Logical Automata

2.1 Definitions

Let us start from some notations of logic. Assume a first-order definite clausal language L is given. We denote by \( L_E \) a definite clausal language especially for describing examples. Let \( f \) be a definite clause in \( L \). In this paper we represent an observation of a dynamic changing world as the sequence of theories.

Definition 1. (Logical Sequence) A logical sequence is defined as

\[
O_0A_0...O_{n-1}A_{n-1}O_n
\]
where $O_i$ is a theory in $L_E$ that describes a set of facts at time $i$. $A_i$ is also the theory in $L_E$ for the action (or output) at time $i$.

A graphical representation of the logical sequence is shown in Fig. 1.

In our models, a state is extended to a logical state where a theory is embedded.

**Definition 2. (Logical State):** A logical state $q$ is a pair $(n, F)$ where $n$ is a natural number for labeling and $F$ is a theory in $L$.

The meaning the logical state $s = (n, F)$ is given by the least Herbrand model of $F$. The logical state could be viewed as a compact state space representation since the least Herbrand model of $F$ is formed in the set of ground facts. We introduce the logical edge as follows:

**Definition 3. (Logical Edge):** A logical edge is an edge between two logical states. A set of logical outputs $E = \{E_1, \ldots, E_m\}$ is embedded in the edge where $E_i$ is a theory.

Next, let us consider conditional state transitions between the logical states. We introduce generality relations between the logical states and the observed facts as well as between the logical edges and the observed actions. Logical entailment is a possible candidate for measuring the relations between theories, however, we employ the subsumption order to avoid the semi-decidability of logical entailment in this paper.

**Definition 4. (Clause Subsumption):** Clause $f_1$ subsumes clause $f_2$, $f_1 \succeq f_2$, if there exists a substitution $\theta$ such that $f_1 \theta \subseteq f_2$.

**Definition 5. (Theory Subsumption [6]):** Theory $F_1$ subsumes theory $F_2$, $F_1 \succeq_T F_2$, iff $\forall C \in F_2$ $\exists C_1 \in F_1$ $C_1 \succeq C_2$.

Under the above theory subsumption order, we define the following conditional state transition between the logical states.

**Definition 6. (Special State Transition (SST)):** For a given partial input sequence $O_i, A_i, O_{i+1}$, the state transition $q' = \sigma(q, E_i)$ between the logical states $q = (n, F)$ and $q' = (n', F')$ occurs iff $F \succeq_T O_i, E_i \succeq_T A_i$ and $F' \succeq_{T} O_{i+1}$.

Intuitively SST accepts the partial input if the input is more special than the related logical states and logical edge. Note that $F$ and $E_i$ are the prior conditions, and $F'$ is the posterior condition in the above state transitions.

The scope of the first-order variables is expanded to the adjacent logical states.

**Definition 7. (Scope of Variables):** For the state transition $q' = \sigma(q, E_i)$ between the logical states $q = (n, F)$ and $q' = (n', F')$, let $V_F, V_E$, and $V_{F'}$ be the sets of the first-order variables that appear in $F$, $E_i$, and $F'$ respectively. Then the scope of the variables are within $V_F \cup V_E \cup V_{F'}$.

Finally we extend probabilistic non-deterministic finite automata (PNFA) to Probabilistic Logical Automata for Special inputs (PLAS) with SST.

**Definition 8. (PLAS):** PLAS is a quintuplet $\langle L, Q, I, \sigma, G \rangle$ where $L$ is a definite language, $Q$ is a finite set of logical states, $I$ is the probability distribution over the initial logical states, $\sigma$ is the special transition probability function from $Q \times L \times Q$ to $2^Q$, and $G$ is the probability distribution over the final logical states.

The brief comparisons of PNFA and PLAS are given in Table 1.
<table>
<thead>
<tr>
<th>PNFA</th>
<th>PLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabet</td>
<td>Definite language</td>
</tr>
<tr>
<td>Word</td>
<td>Logical sequence</td>
</tr>
<tr>
<td>State transition</td>
<td>Special state transition</td>
</tr>
</tbody>
</table>

Table 1. Comparisons of PNFA and PLAS

2.2 Semantics of Logical States

In the previous section, we introduce the generality orders between the logical states and the inputs. Let us consider semantic aspects of PLAS by focusing on the generality orders. Fig.2 shows the inclusion relations between the logical states and logical input sequences in PLAS where \( LM(F) \) is the least model of the theory \( F \), and \( LM(O) \) is the least model of the accepted observation \( O \) in the logical state. This inclusion relation is an example of the following theorem:

**Theorem 1. Logical State in PLAS** In PLAS, if logical state \( S=(n,F) \) accepts the observation \( O_i \) of the logical sequence \( O_0,A_0,...,O_n \), then

\[
LM(O_i) \subseteq LM(F).
\]

**Proof.** \( F \models O_i \) implies \( F \models O_i \). Thus \( LM(O_i) \subseteq LM(F) \) from the definition of the entailment relation.

The observations for PLAS (Fig.2) are the partial observations; PLAS defines the probability distribution over belief states, given a set of the logical sequences.

3 Learning Stochastic Logical Automata

Our next interest is to learn the stochastic logical automata from observations. The above discussion on the semantic aspect of our model indicates that PLAS could be learned by generalising the inputs. Our generalisation algorithms combines logic and probabilistic models through inductive generalisation.

3.1 Setting

**Given:**

- Positive Examples \( E \): A set of Logical Sequences
- Background Knowledge \( BK \): A set of ground atoms

**Learn:**

- PLAS that accept \( E \) by the special state transitions related with \( BK \).

3.2 Overview of the Algorithms

Our inductive generalisation algorithms consist of three parts. The first part describes the graphical generalisation of state transition models. Assume two logical sequences are given as positive examples (Fig.3). These sequences could be viewed as a state transition model. In our algorithm, the state transition models are generalised by applying state mergers. For example, in Fig.3 if node2 and node8 are merged, the node28 in Fig.4 is newly generated. The related transition functions are altered through the generalisation process as shown in Fig.4. Detailed discussions on the state merging technique could be found in [7,8].

The second part involves symbolic generalisation of logic programs which are embedded in each states. Plotkin’s RLGG is used for symbolic generalisation of logic programs. For example, the two logic programs in node2 and node8 (Fig.3) are generalised by RLGG when the two nodes are merged (Fig.4).

The third part covers learning of parameters using statistics derived from the input sequences. The state transitions are unobservable in our settings. The probability distributions over the state transitions and actions are estimated using the EM algorithm. These three steps are iterated until the logical nodes converge to a logical node.
Fig. 3. Positive Examples

Fig. 4. Logical State Merging

4 Example

We implemented our inductive algorithms in a Machine Learning system Cellist that learns PLAS from a set of logical sequences. In order to realise tractability, we put two constraints: the depth bound for the graphical generalisation hypothesis space and the variable depth bound for RLG.

In addition to the scope-extended RLG, Cellist has a function to infer \( is(X, Y, \text{int}) \) atoms that represent \( X = Y + \text{int} \) where \( X \) and \( Y \) are the variables restricted over integers.

4.1 Learning Chemical Reaction Rules from Biochemical Simulator

As an application of the system, we learn chemical reaction rules from StochSim that is a general purpose biochemical simulator in which individual molecules or molecular complexes are represented as individual software objects. Chemical reactions between the molecules occur stochastically according to probabilities derived from the given rate constants.

To run the simulator, a user needs to specify some definitions such as (1) kinds of molecules, (2) the initial numbers of molecules, and (3) possible chemical reaction rules. We assume the following simple chemical reactions in our experiment:

\[
2\text{H}_2 + \text{O}_2 \leftrightarrow 2\text{H}_2\text{O} \\
\text{N}_2 + 3\text{H}_2 \leftrightarrow 2\text{NH}_3
\]

After each simulation, we have a dump file that contains time-series measurements of the concentrations of the molecules. In StochSim, the concentration is represented as the actual number of molecules. For this experiment, we modify the StochSim code to output the one-chemical-reaction time-series results. The following shows two artificial dump files of two simulations:

<table>
<thead>
<tr>
<th>[First Run]</th>
<th>[Second Run]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>h2o</td>
</tr>
<tr>
<td>0.000</td>
<td>2</td>
</tr>
<tr>
<td>1.000</td>
<td>2</td>
</tr>
<tr>
<td>2.000</td>
<td>4</td>
</tr>
<tr>
<td>3.000</td>
<td>4</td>
</tr>
<tr>
<td>4.000</td>
<td>4</td>
</tr>
</tbody>
</table>
where time units are defined as *iterations* at the one-chemical-reaction time-scale. For instance, in the First Run file the number of H2O molecules at Time 0.000 is 2. Since the reactions occur stochastically, the concentrations do not change sometimes. For example, in the above First Run file, the initial concentration (Time 0.000) does not change at the next iteration (Time 1.000). This means that the simulator applied a chemical reaction for the molecules during the iteration, but the reaction did not happen because of its stochastic nature. During Time 1.000 and Time 2.000 in the first run, we could conclude that the reaction

\[ 2\text{H}_2 + \text{O}_2 \rightarrow 2\text{H}_2\text{O} \]

happened, because this is only the reaction that could bring the change in our setting. The outputs are converted into two logical sequences as follows:

\[
\begin{align*}
\{\text{mol}(\text{H}_2,2), \text{mol}(\text{NH}_3,3), \text{mol}(\text{O}_2,2), \text{mol}(\text{H}_2,18), \text{mol}(\text{N}_2,5)\} &\text{ act}(0,0) \\
\{\text{mol}(\text{H}_2,2), \text{mol}(\text{NH}_3,3), \text{mol}(\text{O}_2,2), \text{mol}(\text{H}_2,18), \text{mol}(\text{N}_2,5)\} &\text{ act}(1,0) \\
\{\text{mol}(\text{H}_2,4), \text{mol}(\text{NH}_3,3), \text{mol}(\text{O}_2,1), \text{mol}(\text{H}_2,16), \text{mol}(\text{N}_2,5)\} &\text{ act}(2,0) \\
\{\text{mol}(\text{H}_2,4), \text{mol}(\text{NH}_3,5), \text{mol}(\text{O}_2,1), \text{mol}(\text{H}_2,13), \text{mol}(\text{N}_2,4)\} &\text{ act}(3,0) \\
\{\text{mol}(\text{H}_2,4), \text{mol}(\text{NH}_3,7), \text{mol}(\text{O}_2,1), \text{mol}(\text{H}_2,10), \text{mol}(\text{N}_2,3)\} &
\end{align*}
\]

where \(\text{act}(M,N)\) represents the \(M\)th chemical reaction in the \(N\)th logical sequences.

Our learning task is set as:

**Given:**

- A set of outputs from StochSim in the form of logical sequences.

**Find:**

- Chemical Reaction Rules happened during the simulations.

Note that learning probability distributions is omitted since the number of examples is too small in this example.

The graphical representation of the input is shown in Fig.3. Then Cellist starts to search the hypothesis space by applying the state merging with symbolic generalisation. Since our search space is the version space, the examples are consistent with all hypothesis learned by Cellist. Cellist returns the state transition model shown in Fig.5 with the following is/3 atoms:

\[\text{is}(V_5, V_8, 4), \text{is}(V_3, V_9, 2), \text{is}(V_2, V_5, -3), \text{is}(V_2, V_8, 1), \text{is}(V_1, V_6, -1), \text{is}(V_0, V_1, -1), \text{is}(V_0, V_6, -2)\]

![Fig. 5. Logical State Merging: \(2\text{H}_2 + \text{O}_2 \rightarrow 2\text{H}_2\text{O}\)](image)

For example in Fig.5, \(\text{is}(V_0, V_6, -2)\) represents that in node28 the number of H2O, V0, is changed to V6 in node39 where V0 and V6 has the relation \(V_0 = V_6 - 2\). We can translate the learned rules as:
is(V0,V7,-3), is(V1,V0,1), is(V1,V7,-2), is(V3,V9,-2), is(V4,V8,3), is(V5,V3,3), is(V5,V9,1)

**Fig. 6.** Logical State Merging: N2 + 3H2 => 2NH3

Through the reaction, the number of H2O increases by two, the number of O2 decreases by one, and the number of H2 decreases by 2. The number of N2 and NH3 do not change.

Cellist also returns the model shown in Fig.6 with the following is/3 atoms:

is(V0,V7,-3), is(V1,V0,1), is(V1,V7,-2), is(V3,V9,-2), is(V4,V8,3), is(V5,V3,3), is(V5,V9,1)

We can translate knowledge in node46 and node57 as:

Through the reaction, the number of NH3 increases by two, the number of N2 decreases by one, and the number of H2 decreases by 3. The number of O2 and H2O do not change.

The system learns some additional knowledge in the form of is/3.

## 5 Related Works

PLAS is originally designed as a graphical representation of First-order Stochastic Action Language [9] which is a first-order logical version of dynamic Bayesian networks.

Logical Hidden Markov Model [10] and Logical Markov Decision Programs [11] are closely related to our PLAS model. Logical Decision Program could embed a set of atoms in a state whereas PLAS could put a set of definite clauses in the logical state.

Our logical automata could be viewed as a graphical version of Situation Calculus. Successor state axioms are encoded in our each conditional state transitions.

## 6 Discussions and Conclusions

In this paper, we propose a logical extension of stochastic non-deterministic finite automata. By focusing on the generality orders between the logical states and the partial inputs, the semantic aspects of our models become clear; PLAS defines probability distributions over belief states. We also proposed inductive learning algorithms of PLAS by combining graphical and symbolic generalisations smoothly. Parameter learning of PLAS are implemented using EM algorithm in our Machine Learning system “Cellist”.

Since PLAS defines probability distributions over belief states, PLAS would be suitable for representing POMDPs. Note that our induction algorithm could combine the observations and unobserved facts together if the unobserved facts are encoded in background knowledge.

Our logical sequence represents temporal changes of theories, that is, we could input a series of Logic Programs (LPs) to our model. If we learn a PLAS model from the observed series of LPs, the model should capture how LPs change proceeding with time. In [12], a STRIPS-like first-order stochastic operator is proposed in order to modify LPs temporally. The operator is expressed in the form of dynamic Bayesian networks with add/delete-lists functions. The development of the translation algorithms between the operator and PLAS would be useful for the distributed executions of LPs in the Multi-Agent research where the knowledge of an agent could be represented in a LP and he/she modifies the LP by interacting his/her surrounding environments.

Since our models are based on automata, further extensions would be possible by considering the existing extensions of automata such as pushdown automata.
References