Learning from Interpretation Transition

Katsumi Inoue
National Institute of Informatics, Japan
Chiaki Sakama
Wakayama University, Japan

ILP 2012

Dubrovnik, September 18th, 2012

Learning Dynamics of Systems

- Learning action theories in ILP
  - Event calculus: Moyle & Muggleton (1997), Moyle (2003)
  - Logic programs: with situation calculus: Otero (2003, 2005)
  - Action languages: Inoue *et al.* (2005), Tran & Baral (2009)
  - Probabilistic logic programs: Corapi *et al.* (2011)
- Abductive action learning
  - Abductive event calculus: Eshghi (1988), Shanahan (2000)
- Active learning of action models
  - STRIPS-like: Rodrigues *et al.* (2011)

- These works suppose applications to robotics and bioinformatics.
- However, it is hard to infer *rules of systems dynamics* due to presence of positive and negative feedbacks.

LFIT: Learning from Interpretation Transitions

- Herbrand interpretation I: a state of the world
- Logic program P: a *state transition system*, which maps an Herbrand interpretation into another interpretation (Blair *et al.*, 1995—1997; Inoue, 2011; Inoue & Sakama, 2012)
- Next state $T_P(I)$: where T_P is the immediate consequence operator (T_P *operator*).
- We propose a new learning setting in ILP:
  - Given: a set of pairs of Herbrand interpretations (I,J) such that $J = T_P(I)$,
  - Induce a program P.
- C.f. learning from interpretations (LFI)
  - Given: a set S of Herbrand interpretations,
  - Induce a program P whose models are exactly S.

LFIT Applied to Dynamic Systems

- Learning rules of dynamic systems
  - Cellular Automata (CAs): mathematical model of complex adaptive systems (Conway, Wolfram)
  - Boolean Networks (BNs): logical model of gene regulation networks (Kauffman)
- CAs and BNs can be characterized as logic programs, and T_P operator captures their synchronous update (Inoue 2011).

- A learned program P is a *normal logic program* (NLP) in this case.
- Learning NLPs has been considered in ILP, but most approaches take the setting of *learning from entailment*.
- Learning NLPs under the *supported model semantics*.

Normal Logic Programs and T_P operator

- A *normal logic program* (NLP) *P* is a set of rules:

$$H \leftarrow A_1 \land \ldots \land A_m \land \neg B_1 \land \ldots \land \neg B_n \quad (m,n \geq 0)$$

  where H, A_i and B_j are atoms and $\neg$ is (*default*) *negation.*
- *ground*(*P*) : the set of ground instances of all rules in *P*.
- The *Herbrand base* **B** is the set of ground atoms from language(*P*).
- An (*Herbrand*) *interpretation* *I* (of *P*) is a subset of **B**.
- $T_P(I)$:= $\{ H \mid H \leftarrow L_1 \land \ldots \land L_n \in ground(P),\ I \models L_1 \land \ldots \land L_n \}$.
- When *P* is a *definite* program, T_P operator is *monotone*, and $T_P \uparrow \omega$ is the *least model* of *P* (van Emden & Kowalski, 1976).
- When *P* is a *normal* program, T_P is *nonmonotone* (Apt *et al*., 1988).
- The *orbit* of *I* wrt P (Blair *et al*., 1997) is $\langle T_P^k(I) \rangle_{k=0,1,2,\ldots}$, where $T_P^0(I) = I$, $T_P^{k+1}(I) = T_P(T_P^k(I))$ for $k = 0, 1, 2, \ldots$.

Supported Models / Supported Classes

- An interpretation I is ***supported*** (Apt, Blair & Walker, 1988) if $\forall A \in I.\ \exists (A \leftarrow A_1 \wedge \ldots \wedge A_m \wedge \neg B_1 \wedge \ldots \wedge \neg B_n) \in ground(P)$ such that $\forall i.\ A_i \in I$ and $\forall j.\ B_j \notin I$.
- **Prop.** I is a **supported model** of P iff $I = T_P(I)$.
- A **supported class** (Inoue & Sakama, 2012) of an NLP P is a nonempty set $\boldsymbol{S}$ of Herbrand interpretations satisfying

$$\boldsymbol{S} = \{\ T_P(I) \mid I \in \boldsymbol{S}\ \}.$$

- A supported class $\boldsymbol{S}$ of P is ***strict*** if no proper subset of $\boldsymbol{S}$ is a supported class of P.
- **Theorem** (Inoue & Sakama, 2012): A finite set $\boldsymbol{S}$ of Herbrand interpretations is a strict supported class of P iff there is a directed cycle $I_1 \rightarrow I_2 \rightarrow \ldots \rightarrow I_k \rightarrow I_1$ ($k \geq 1$) in the state transition graph induced by T_P such that $\{I_1, I_2, \ldots., I_k\} = \boldsymbol{S}$.

Supported Classes = Attractors

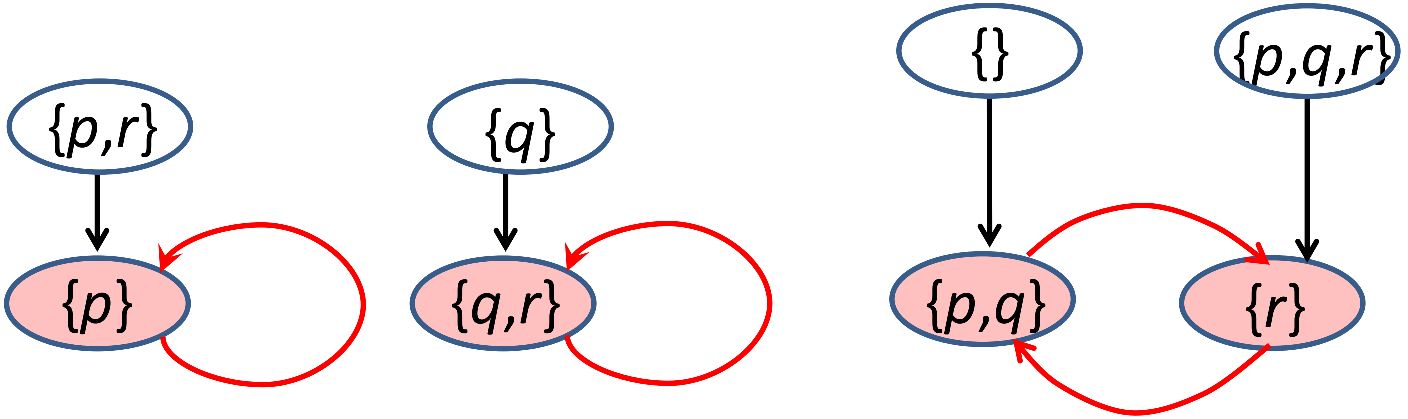
- P_1:

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

- There are 3 strict supported classes of P_1:

$$S_1 = \{\{p\}\},\ S_2 = \{\{q, r\}\},\ S_3 = \{\{p, q\}, \{r\}\}.$$

- S_1 and S_2 are the supported models of P_1 (*point attractors*).



LF1T: **L**earning **f**rom **1**-Step **T**ransitions

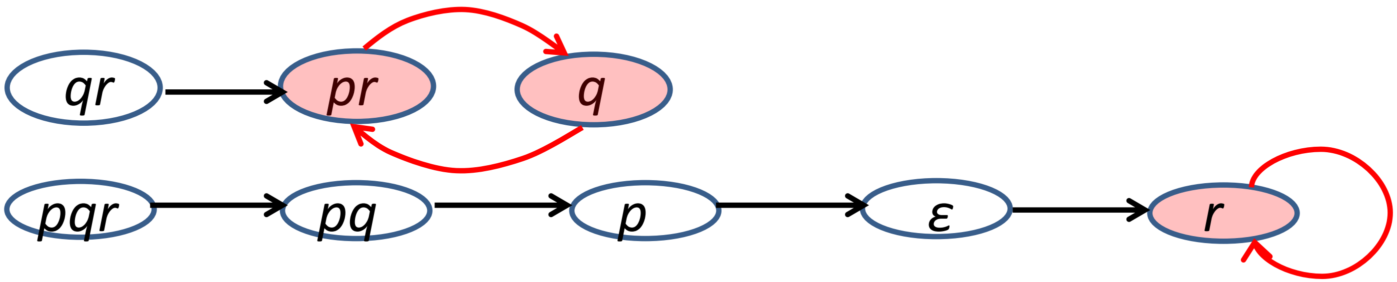
- **Input:** $E \subseteq 2^{\mathbf{B}} \times 2^{\mathbf{B}}$: (positive) examples/observations
- **Output:** NLP P s.t. $J = T_P(I)$ holds for any $(I, J) \in E$

1. If $E = \emptyset$, then output P and stop;
2. Pick $(I, J) \in E$; put $E := E \setminus \{(I, J)\}$;
3. For each $A \in J$, let $R^I_A := A \leftarrow \bigwedge_{B \in I} B \wedge \bigwedge_{C \in \mathbf{B} \setminus I} \neg C$;
4. Update $P :=$ **AddRule**(R^I_A, 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';
3. Find a rule $R' \in P \cup P'$ s.t. $h(R) = h(R')$ and $b(R')$ and $b(R)$ differ in the sign of only one literal. If there is no such a rule in P, return $P \cup \{R\}$;
4. Return **AddRule**$(lg(R,R'), P \setminus \{R'\})$, where lg is the least generalization.

LF1T: Example [$R^{I}_{A} := A \leftarrow \bigwedge_{B \in I} B \wedge \bigwedge_{C \in \mathbf{B} \setminus I} \neg C$]



Step	$I \rightarrow J$	Operation	Rule	ID	P	P'
1	$qr \rightarrow pr$	R^{qr}_{p}	$p \leftarrow \neg p \wedge q \wedge r$	1	1	{}
		R^{qr}_{r}	$r \leftarrow \neg p \wedge q \wedge r$	2	1,2	
2	$pr \rightarrow q$	R^{pr}_{q}	$q \leftarrow p \wedge \neg q \wedge r$	3	1,2,3	
3	$q \rightarrow pr$	R^{q}_{p}	$p \leftarrow \neg p \wedge q \wedge \neg r$	4		
		$lg(4,1)$	$p \leftarrow \neg p \wedge q$	5	2,3,5	+1,4
		R^{q}_{r}	$r \leftarrow \neg p \wedge q \wedge \neg r$	6		
		$lg(6,2)$	$r \leftarrow \neg p \wedge q$	7	3,5,7	+2,6
4	$pqr \rightarrow pq$	R^{pqr}_{p}	$p \leftarrow p \wedge q \wedge r$	8		
		$lg(8,1)$	$p \leftarrow q \wedge r$	9	3,5,7,9	+8
		R^{pqr}_{q}	$q \leftarrow p \wedge q \wedge r$	10		
		$lg(10,3)$	$q \leftarrow p \wedge r$	11	5,7,9,11	+3,10

Example (cont.) $[R^I_A := A \leftarrow \bigwedge_{B\in I} B \wedge \bigwedge_{C\in \mathbf{B}\setminus I} \neg C]$

Step	$I \rightarrow J$	Operation	Rule	ID	P	P'
5	$pq \rightarrow p$	R^{pq}_p	$p \leftarrow p \wedge q \wedge \neg r$	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	$p \rightarrow \varepsilon$					
7	$\varepsilon \rightarrow r$	R^{ε}_r	$r \leftarrow \neg p \wedge \neg q \wedge \neg r$	15		
		$lg(15,6)$	$r \leftarrow \neg p \wedge \neg r$	16	7,11,14,16	+15
8	$r \rightarrow r$	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

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

propositional program

$$p(t+1) \leftarrow q(t).$$
$$q(t+1) \leftarrow p(t) \wedge q(t).$$
$$r(t+1) \leftarrow \neg p(t).$$

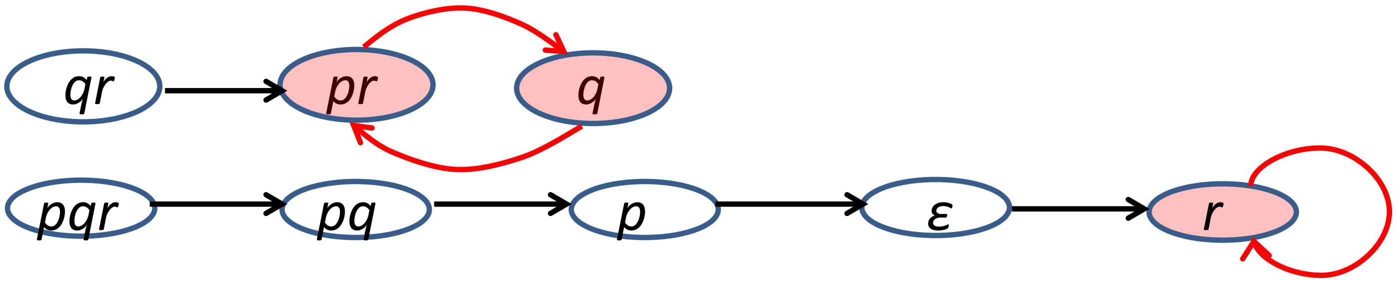
first-order program

LFBA: **L**earning **f**rom **B**asins of **A**ttraction

- **Input:** $\mathcal{E} \subseteq 2^{2^{\mathbf{B}}}$: (positive) examples/observations
- **Output:** NLP P s.t. for $\forall \mathcal{I} \in \mathcal{E}$, any $I \in \mathcal{I}$ belongs to the basin of attraction of some attractor of P contained in $\mathcal{I}$
- **Assumption:** Each $\mathcal{I}$ contains the interpretations belonging to the orbit of some $I_0 \in \mathcal{I}$ wrt T_P, and that $\mathcal{I}$ constitutes a sequence $I_0 \rightarrow I_1 \rightarrow \dots \rightarrow I_{k-1} \rightarrow J_0 \rightarrow \dots \rightarrow J_{l-1} \rightarrow J_0 \rightarrow \dots$, where $|\mathcal{I}| = k + l$ and $\{J_0, \dots, J_{l-1}\}$ is an attractor.
- 2 orbits $\mathcal{I}, \mathcal{J} \in \mathcal{E}$ reach the same attractor iff $\mathcal{I} \cap \mathcal{J} = \emptyset$.

1. Put $P := \emptyset$; $P' := \emptyset$;
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 $E := \{(I, J) \mid I, J \in \mathcal{I},\ J$ is the next state of $I\}$;
5. $P :=$ **LF1T**(E, P, P'); Return to 2.

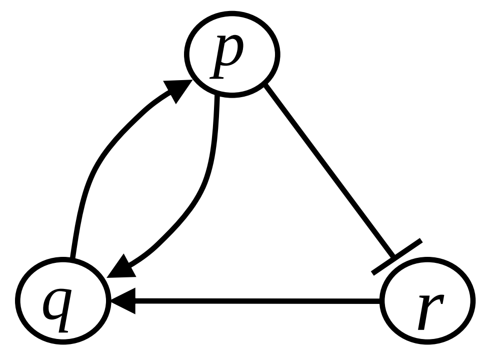
LFBA: Example



Input: $\mathcal{E} = \{I_1, I_2\}$

I_1 : $qr \rightarrow pr \rightarrow q \rightarrow pr \rightarrow q \rightarrow \ldots$

I_2 : $pqr \rightarrow pq \rightarrow p \rightarrow \varepsilon \rightarrow r \rightarrow r \rightarrow \ldots$



LF1T(E_1, Ø, Ø) = {3,5,7};

LF1T(E_2, {3,5,7}, {1,2,4,6}) = {11,14,19};

In general, identification of an exact NLP using **LF1T** may require $2^{|\mathbf{B}|}$ examples, while $|\mathcal{E}|$ in **LFBA** is bounded by $c\delta$, where δ is the number of attractors.

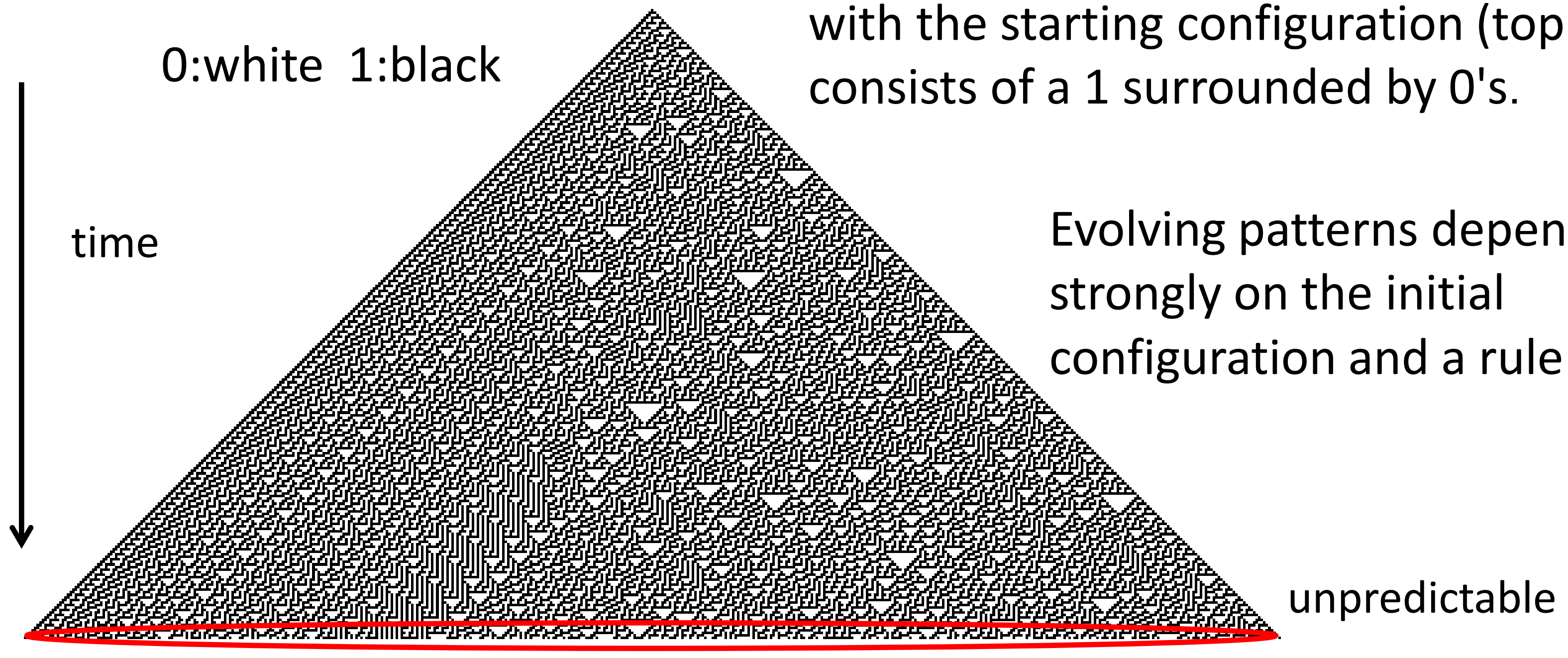
Cellular Automata (CA)

- A CA consists of a regular grid of cells.
- A cell has a finite number of possible states.
- The state of each cell changes synchronously in discrete time steps according to local and identical transition rules.
- The state of a cell in the next time step is determined by its current state and the states of its surrounding cells (neighborhood).
- CA is a model of *emergence* and *self-organization*, which are two important features of the nature (the real life) as a complex system.
- 1-dimensional 2-state CA can simulate Turing Machine (Wolfram).
- 2-dimensional 2-state CA is known as LIFE (Conway).
- 2-state CA is an instance of Boolean networks.

1-Dimensional CA

current pattern	111	110	101	100	011	010	001	000
new state for center cell	0	0	0	1	1	1	1	0

Wolfram's Rule 30



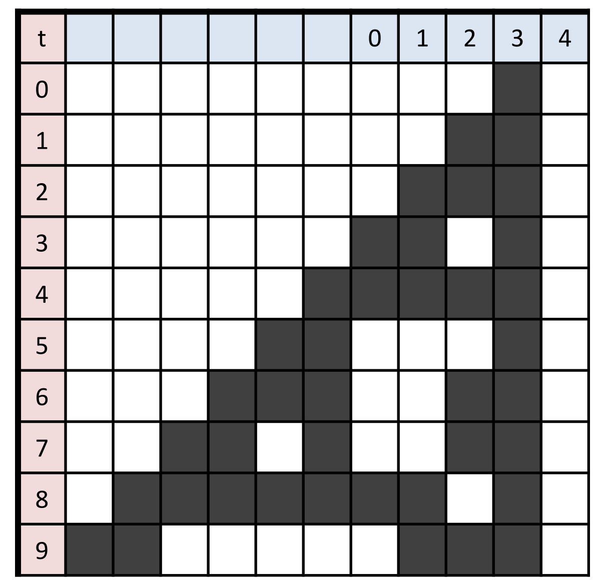
The history of the generated patterns with the starting configuration (top) that consists of a 1 surrounded by 0's.

Evolving patterns depend strongly on the initial configuration and a rule used.

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

- $c(x,t{+}1) \leftarrow \neg c(x\text{-}1,t) \wedge \neg c(x,t) \wedge c(x{+}1,t).$
- $c(x,t{+}1) \leftarrow \neg c(x\text{-}1,t) \wedge c(x,t) \wedge \neg c(x{+}1,t).$
- $c(x,t{+}1) \leftarrow \neg c(x\text{-}1,t) \wedge c(x,t) \wedge c(x{+}1,t).$
- $c(x,t{+}1) \leftarrow \neg c(x\text{-}1,t) \wedge c(x,t) \wedge \neg c(x{+}1,t).$
- $c(x,t{+}1) \leftarrow c(x\text{-}1,t) \wedge \neg c(x,t) \wedge c(x{+}1,t).$

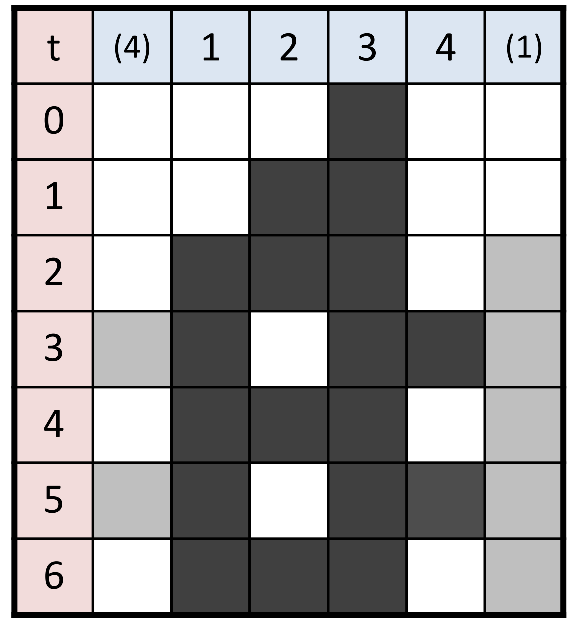


- Rule 110 is known to be Turing-complete.
- The logic program is *acyclic* (Apt & Bezem, 1990).

Incorporating Background Theories

- Torus world: length 4
- c(0, t) ← c(4, t).
- c(5, t) ← c(1, t).

c(3)
→ c(2), c(3)
→ c(1), c(2), c(3)
→ c(1), c(3), c(4) attractor
→ c(1), c(2), c(3) → …



learning rules: 0→1 (4), 1→2 (2), 2→3 (2).

learning positive rules: (2), (2), (1).

Incorporating Inductive Bias I

- Bias I: The body of each rule exactly contains 3 neighbor literals.

Step	$I \rightarrow J$	Op.	Rule	ID	P
1	0010→0110	$R^3{}_2$	$c(2) \leftarrow \neg c(1) \wedge \neg c(2) \wedge c(3)$	1	1
		$R^3{}_3$	$c(3) \leftarrow \neg c(2) \wedge c(3) \wedge \neg c(4)$	2	1,2
2	0110→1110	$R^2{}_1$	$c(1) \leftarrow \neg c(0) \wedge \neg c(1) \wedge c(2)$	3	
		$lg(3,1)$	$c(x) \leftarrow \neg c(x\text{-}1) \wedge \neg c(x) \wedge c(x+1)$	4	2,4
		$R^{23}{}_2$	$c(2) \leftarrow \neg c(1) \wedge c(2) \wedge c(3)$	5	2,4,5
		$R^{23}{}_3$	$c(3) \leftarrow c(2) \wedge c(3) \wedge \neg c(4)$	6	2,4,5,6
3	1110→1011	$R^{12}{}_1$	$c(1) \leftarrow \neg c(0) \wedge c(1) \wedge c(2)$	7	
		$lg(7,5)$	$c(x) \leftarrow \neg c(x\text{-}1) \wedge c(x) \wedge c(x+1)$	8	2,4,6,8
		$R^{34}{}_4$	$c(4) \leftarrow c(3) \wedge \neg c(4) \wedge c(5)$	9	2,4,6,8,9
4	1011→1110	$R^{01}{}_1$	$c(1) \leftarrow c(0) \wedge c(1) \wedge \neg c(2)$	10	
		$lg(10,6)$	$c(x) \leftarrow c(x\text{-}1) \wedge c(x) \wedge \neg c(x+1)$	11	2,4,8,9,11

Incorporating Inductive Bias II

- Bias II: The rules are universal for every time step.
- Biases I and II imply that *anti-instantiation* (AI) can be applied immediately instead of least generalization.

Step	$I \rightarrow J$	Op.	Rule	ID	P
1	0010→0110	R^3_2	$c(2) \leftarrow \neg c(1) \wedge \neg c(2) \wedge c(3)$	1	
		AI(1)	$c(x) \leftarrow \neg c(x\text{-}1) \wedge \neg c(x) \wedge c(x+1)$	2	2
		R^3_3	$c(3) \leftarrow \neg c(2) \wedge c(3) \wedge \neg c(4)$	3	
		AI(3)	$c(x) \leftarrow \neg c(x\text{-}1) \wedge c(x) \wedge \neg c(x+1)$	4	2,4
2	0110→1110	R^{23}_2	$c(2) \leftarrow \neg c(1) \wedge c(2) \wedge c(3)$	5	
		AI(5)	$c(x) \leftarrow \neg c(x\text{-}1) \wedge c(x) \wedge c(x+1)$	6	2,4,6
		R^{23}_3	$c(3) \leftarrow c(2) \wedge c(3) \wedge \neg c(4)$	7	
		AI(7)	$c(x) \leftarrow c(x\text{-}1) \wedge c(x) \wedge \neg c(x+1)$	8	2,4,6,8
3	1110→1011	R^{34}_4	$c(4) \leftarrow c(3) \wedge \neg c(4) \wedge c(5)$	9	
		AI(9)	$c(x) \leftarrow c(x\text{-}1) \wedge \neg c(x) \wedge c(x+1)$	10	2,4,6,8,10

Conclusion & Future Work

- Learning complex networks becomes more and more important.
- We tackled the induction problem of such dynamic systems in terms of NLP learning from synchronous state transitions.
- A more efficient construction in the bottom-up algorithm.
- More complex schemes such as asynchronous and probabilistic updates do not obey transition by the T_P operator.
- Applications to large and multi-state CAs.