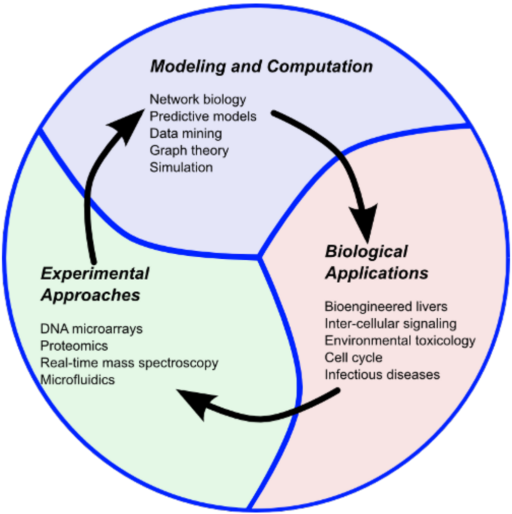
Using Reachability Properties of Logic Program for Revising Biological Models

Xinwei Chai, *Tony Ribeiro*, Morgan Magnin, Olivier Roux, Katsumi Inoue

Laboratoire des Sciences du Numérique de Nantes, France
National Institute of Informatics, Tokyo

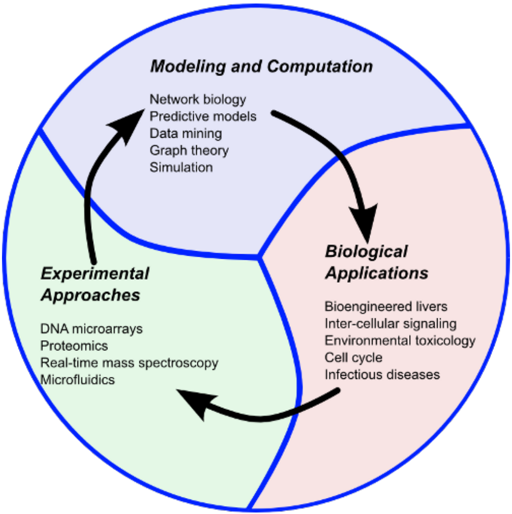
September 4, 2018

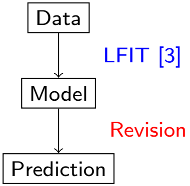
Outline



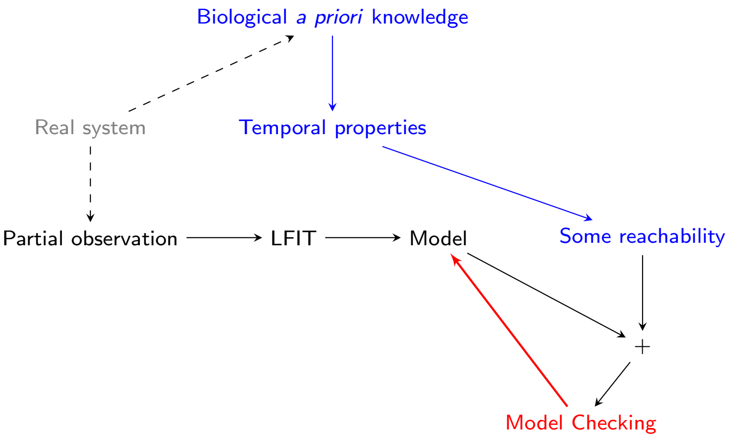
Modeling and Computation
Network biology
Predictive models
Data mining
Graph theory
Simulation
Biological Applications
Bioengineered livers
Inter-cellular signaling
Environmental toxicology
Cell cycle
Infectious diseases
Experimental Approaches
DNA microarrays
Proteomics
Real-time mass spectroscopy
Microfluidics

Outline


Modeling and Computation
Network biology
Predictive models
Data mining
Graph theory
Simulation
Biological Applications
Bioengineered livers
Inter-cellular signaling
Environmental toxicology
Cell cycle
Infectious diseases
Experimental Approaches
DNA microarrays
Proteomics
Real-time mass spectroscopy
Microfluidics


Data
LFIT [3]
Model
Revision
Prediction

Process Scheme



Modelings

Boolean Network

$$f(a) = \neg b$$
$$f(b) = a$$

$$\iff$$

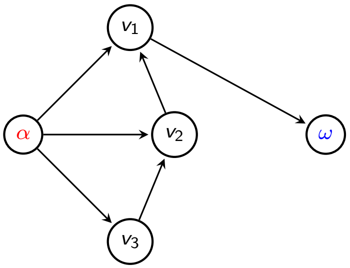
Logic Program

$$a(t+1) \leftarrow \neg b(t)$$
$$b(t+1) \leftarrow a(t)$$

$$\begin{tikzcd}
(0,0) \arrow[r] & (1,0) \arrow[d] \\
(0,1) \arrow[u] & (1,1) \arrow[l]
\end{tikzcd}$$

State transition graph

Reachability problem



Given a BN, from initial state α, does there exist a transition sequence that reaches the target state ω?

$\Updownarrow$

Given a state transition graph, from initial state α, does there exist a pathway towards the target state ω?

Reachability of global states $\boxed{\mathbf{EF}(a_i, b_j, \ldots)}$ $\rightarrow$ computationally difficult

$\Longrightarrow$ Reachability of local states $\boxed{\mathbf{EF}\, a_i}$

Difficulties and solution

- State space grows exponentially with the number of automata
- Traditional model checkers e.g. Mole[1] and NuSMV[2] fail
  global search → time out and/or out of memory
- **Static analysis**: avoid global search, at the cost of precision

→ A balance between time-space performance and conclusiveness

- Paulevé *et al.* introduced LCG (Local Causality Graph) [1, 2] for static analysis
- Implementation: Pint
- Efficient (beats many traditional model checkers) **but**
- Usually not conclusive when the density of the biological network increases

[1] http://www.lsv.fr/~schwoon/tools/mole
[2] http://nusmv.fbk.eu

Local Causality Graph (LCG)

Start with target state $\omega \rightarrow$ Find transitions reaching $\omega \rightarrow$ Find new target states to fire those transitions $\rightarrow \cdots$ Recursion $\cdots \rightarrow$ End with initial state α

- Goal-oriented structure
- Formed by recursive updates
- Avoid global search in state transition graphs

Example of LCG

Initial state $\alpha = \langle a_0, b_1, c_0, d_0, e_0 \rangle$, target state $\omega = a_1$

Rules: $a_1 \leftarrow b_1 \wedge c_1$, $a_1 \leftarrow e_1$,
$b_1 \leftarrow d_0$, $c_1 \leftarrow d_1$, $d_1 \leftarrow b_1$

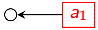
$\boxed{a_1}$

Small circles stand for transition nodes, squares for state nodes

Example of LCG

Initial state $\alpha = \langle a_0, b_1, c_0, d_0, e_0 \rangle$, target state $\omega = a_1$

Rules: $a_1 \leftarrow b_1 \wedge c_1$, $a_1 \leftarrow e_1$,
$b_1 \leftarrow d_0$, $c_1 \leftarrow d_1$, $d_1 \leftarrow b_1$

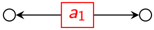


Small circles stand for transition nodes, squares for state nodes

Example of LCG

Initial state $\alpha = \langle a_0, b_1, c_0, d_0, e_0 \rangle$, target state $\omega = a_1$

Rules: $a_1 \leftarrow b_1 \wedge c_1$, $a_1 \leftarrow e_1$,
$b_1 \leftarrow d_0$, $c_1 \leftarrow d_1$, $d_1 \leftarrow b_1$

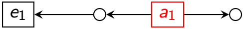


Small circles stand for transition nodes, squares for state nodes

Example of LCG

Initial state $\alpha = \langle a_0, b_1, c_0, d_0, e_0 \rangle$, target state $\omega = a_1$

Rules: $a_1 \leftarrow b_1 \land c_1$, $a_1 \leftarrow e_1$,
$b_1 \leftarrow d_0$, $c_1 \leftarrow d_1$, $d_1 \leftarrow b_1$

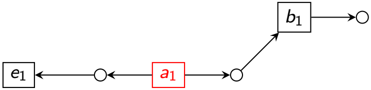


Small circles stand for transition nodes, squares for state nodes

Example of LCG

Initial state $\alpha = \langle a_0, b_1, c_0, d_0, e_0 \rangle$, target state $\omega = a_1$

Rules: $a_1 \leftarrow b_1 \wedge c_1$, $a_1 \leftarrow e_1$,
$b_1 \leftarrow d_0$, $c_1 \leftarrow d_1$, $d_1 \leftarrow b_1$

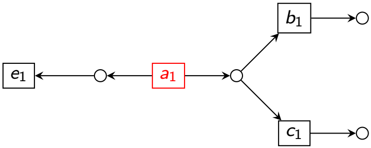


Small circles stand for transition nodes, squares for state nodes

Example of LCG

Initial state $\alpha = \langle a_0, b_1, c_0, d_0, e_0 \rangle$, target state $\omega = a_1$

Rules: $a_1 \leftarrow b_1 \wedge c_1$, $a_1 \leftarrow e_1$,
$b_1 \leftarrow d_0$, $c_1 \leftarrow d_1$, $d_1 \leftarrow b_1$

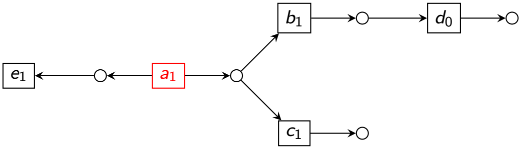


Small circles stand for transition nodes, squares for state nodes
$r'(a_1) = r'(e_1) \vee (r'(b_1) \wedge r'(c_1))$

Example of LCG

Initial state $\alpha = \langle a_0, b_1, c_0, d_0, e_0 \rangle$, target state $\omega = a_1$

Rules: $a_1 \leftarrow b_1 \wedge c_1$, $a_1 \leftarrow e_1$,
$b_1 \leftarrow d_0$, $c_1 \leftarrow d_1$, $d_1 \leftarrow b_1$

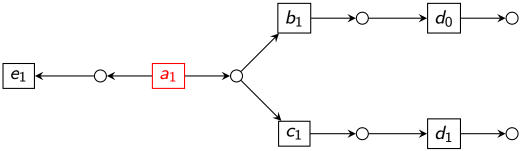


Small circles stand for transition nodes, squares for state nodes
$r'(a_1) = r'(d_0) \wedge r'(c_1)$

Example of LCG

Initial state $\alpha = \langle a_0, b_1, c_0, d_0, e_0 \rangle$, target state $\omega = a_1$

Rules: $a_1 \leftarrow b_1 \wedge c_1$, $a_1 \leftarrow e_1$,
$b_1 \leftarrow d_0$, $c_1 \leftarrow d_1$, $d_1 \leftarrow b_1$

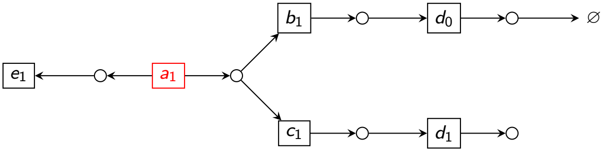


Small circles stand for transition nodes, squares for state nodes
$r'(a_1) = r'(d_0) \wedge r'(d_1)$

Example of LCG

Initial state $\alpha = \langle a_0, b_1, c_0, d_0, e_0 \rangle$, target state $\omega = a_1$

Rules: $a_1 \leftarrow b_1 \wedge c_1$, $a_1 \leftarrow e_1$,
$b_1 \leftarrow d_0$, $c_1 \leftarrow d_1$, $d_1 \leftarrow b_1$

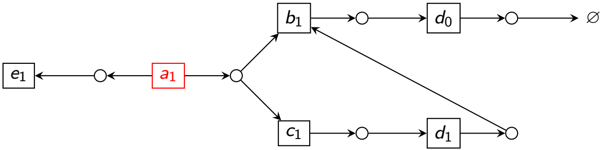


Small circles stand for transition nodes, squares for state nodes
$r'(a_1) = r'(d_1)$

Example of LCG

Initial state $\alpha = \langle a_0, b_1, c_0, d_0, e_0 \rangle$, target state $\omega = a_1$

Rules: $a_1 \leftarrow b_1 \wedge c_1$, $a_1 \leftarrow e_1$,
$b_1 \leftarrow d_0$, $c_1 \leftarrow d_1$, $d_1 \leftarrow b_1$

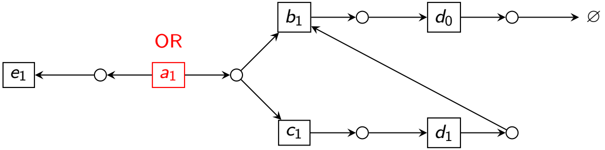


Small circles stand for transition nodes, squares for state nodes
$r'(a_1) = r'(b_1) = r'(d_0) = 1$

Example of LCG

Initial state $\alpha = \langle a_0, b_1, c_0, d_0, e_0 \rangle$, target state $\omega = a_1$

Rules: $a_1 \leftarrow b_1 \wedge c_1$, $a_1 \leftarrow e_1$,
$b_1 \leftarrow d_0$, $c_1 \leftarrow d_1$, $d_1 \leftarrow b_1$

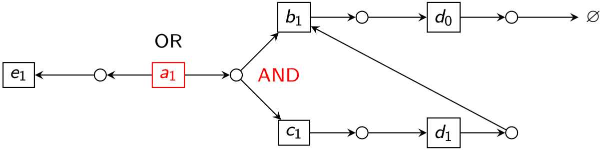


Small circles stand for transition nodes, squares for state nodes

Example of LCG

Initial state $\alpha = \langle a_0, b_1, c_0, d_0, e_0 \rangle$, target state $\omega = a_1$

Rules: $a_1 \leftarrow b_1 \wedge c_1,\ a_1 \leftarrow e_1,$
$b_1 \leftarrow d_0,\ c_1 \leftarrow d_1,\ d_1 \leftarrow b_1$



Small circles stand for transition nodes, squares for state nodes

Algorithm for Reachability

- Input: A logic program P, an initial state α, a target state ω and a max number of iterations k
- Output: $reach(\omega) \in \{\textbf{False}, \textbf{True}, \textbf{Inconclusive}\}$

1. Construct the LCG $\ell = LCG(P, \alpha, \omega)$
2. Try to remove all cycles and prune useless edges from ℓ
3. Try to prove unreachability of ω in ℓ using pseudo-reachability $reach'(\ell, \omega)$ and return **False** if $reach'(\ell, \omega) = \textbf{False}$
4. Try at most k times
   - $\ell' \leftarrow \ell$
   - Simplify each **OR gate** such that ℓ' is a LCG with only **AND gates**
   - If there remain cycles:
     - Back to step (4)
   - Generate all trajectory that starts with α in ℓ' using ASP
     - If a trajectory t ending with ω is found, return **True**
5. return **Inconclusive**

ASPReach

In an LCG, link $a_1 \rightarrow \circ \rightarrow b_1$ can be translated as:
`node('a','1',1).  node('b','1',2).  parent(1,2).`

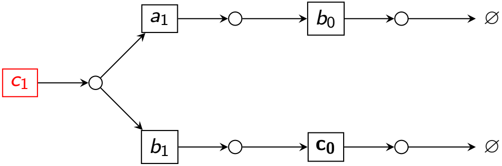
Core code:

```
prior(N1,N2) :-   parent(N2,N1).  %Rule 1
prior(N1,N3) :-   prior(N1,N2), prior(N2,N3).  %Rule 2
prior(N1,N2) :-   node(P1,S1,N1), node(P2,S2,N2),
                  node(P2,S3,N3), parent(N1,N3),
                  init(P2,S3), S2!=S3, P1!=P2.  %Rule 3
```

`N` for node, `P` for component, `S` for state
Rule 3: in the LCG, one branch contains $a_1 \rightarrow \circ \rightarrow b_0$, another branch contains b_1, if $b_0 \in \alpha$, a_1 is to be reached before reaching b_1

Example

Initial state $\alpha = a_0, b_0, c_0$, target state $\omega = c_1$
Rules: $a_1 \leftarrow b_0, b_1 \leftarrow c_0, c_1 \leftarrow a_1 \wedge b_1$



$a \rhd b$ means a appears in the sequence before b

Rule 1 & 2 $\Rightarrow$ $b_0 \rhd a_1 \rhd c_1$, $c_0 \rhd b_1 \rhd c_1$

Rule 3 $\Rightarrow$ $a_1 \rhd b_1$

The only admissible order is $a_1 \rightarrow b_1 \rightarrow c_1$

Benchmark

Traditional model checkers: Mole NuSMV $\rightarrow$ **memory-out**
Pure static analyzer: Pint [1]
Small example: λ-phage, 4 components
Big examples: TCR (T-Cell Receptor, 95 components) and
EGFR (Epidermal Growth Factor Receptor, 106 components)

Model	λ-phage			TCR			EGFR		
Inputs	4			3			13		
Outputs	4			5			12		
Total tests	$2^4 \times 4 = 64$			$2^3 \times 5 = 40$			$2^{13} \times 12 = 98,304$		
Analyzer	Pint	**PR**	**AR**	Pint	**PR**	**AR**	Pint	**PR**	**AR**
Reachable	36(56%)	38(59%)	38(59%)	16(40%)			64,282(65.4%)	74,268(75.5%)	
Inconclusive	**2(3%)**	**0(0%)**		0(0%)			**9,986(10.1%)**	**0(0%)**	
Unreachable	26(41%)			24(60%)			24,036(24.5%)		
Total time	$<$ 1s			7s	0.85s	40s	**9h50min**	**15min31s**	**3h46min**

PR=PermReach, AR=ASPReach

Collaboration with LFIT

- If the model is consistent with *a priori* knowledge
  - Do nothing
- If not consistent

	Reachable	Unreachable
Knowledge	R_K	U_K
Inferred model	R_I	U_I
Inconsistency (problem)	$R'_K = R_K \cap U_I$	$U'_K = R_I \cap U_K$
Keep consistent with	U_K	R_K
Operation	Generalization◯	Specialization◯
	Add transitions×	Delete transitions◯

where set R and U are consisting of pairs of form (α, ω)

Definitions

Specialization of a transition

By adding elements in the body of a transition, it is possible to change a reachable state to an unreachable one

Generalization of a transition

By deleting elements in the body of a transition, it is possible to change an unreachable state to a reachable one

Main Algorithm

- Input: an Automata Network A, reachable set R_K, unreachable set U_K
- Output: modified Automata Network A or $\varnothing$ if not revisable

1. Construct the LCGs for the elements in R_K and U_K, collect inconsistent instances in set R'_K and U'_K
2. Specialize the transitions to make elements in U'_K unreachable, if not possible, return $\varnothing$
3. Generalize the transitions to make elements in R'_K reachable, if not possible, return $\varnothing$
4. Return A

Specialization

- Input: a logic program P, an unsatisfied element (α, ω), a reachable set Re, an unreachable set Un
- Output: modified logic program P or $\emptyset$ if not revisable

1. $Rev \leftarrow \{\omega\}$
2. For each R s.t. $h(R) = Rev$, for each $R' \in \{R'' | R'' \in ls(R) \wedge \nexists (I, J) \in E$, s.t. $\nexists R''' \in P \cup \{R''\} \setminus \{R\}, h(R''') \in J, b(R''') \in I\}$
   - If $P' \leftarrow P \setminus \{R\} \cup \{R'\}$, $unreachable(P', \alpha, \omega)$ and P' satisfies all previous properties, return P'
3. $Rev \leftarrow b(R)$ with $h(R) = Rev$ and back to step 2
4. There is no revision for (α, ω), return $\emptyset$

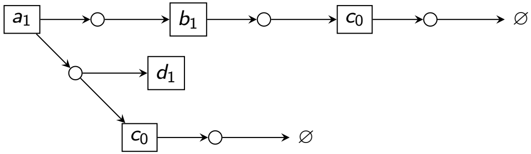
Generalization

- Input: a logic program P, an unsatisfied element (α, ω), a reachable set Re, an unreachable set Un
- Output: modified logic program P or $\emptyset$ if not revisable

1. $Rev \leftarrow \{\omega\}$
2. For each R s.t. $h(R) = Rev$, for each $R' \in lg(R)$
   - If $P' \leftarrow P \setminus \{R\} \cup \{R'\}$, $reachable(P', \alpha, \omega)$ and P' satisfies all previous properties, return P'
3. $Rev \leftarrow b(R)$ with $h(R) = Rev$ and back to step 2
4. There is no revision for (α, ω), return $\emptyset$

Example

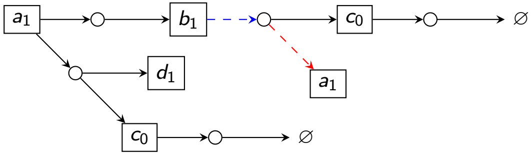
Rules: $a_1 \leftarrow b_1,\ a_1 \leftarrow d_1 \wedge c_0,\ b_1 \leftarrow c_0,\ c_1 \leftarrow b_0$
Initial state: $\alpha = \langle a_0, b_0, c_0, d_0 \rangle$
$U_K = \{(\alpha, b_1), (\alpha, d_1)\},\ R_K = \{(\alpha, a_1)\}$



- $L = \{\{(\alpha, a_1), (\alpha, b_1), (\alpha, d_1)\}, \{(\alpha, b_1)\}, \{(\alpha, d_1)\}\}$
- Start from $\{(\alpha, b_1)\}$ and $\{(\alpha, d_1)\}$
- $b_1 \leftarrow c_0$ can be specialized to $b_1 \leftarrow c_0 \wedge a_1$ to make b_1 unreachable
- $a_1 \leftarrow d_1 \wedge c_0$ can only be generalized to $a_1 \leftarrow c_0$ as $d_1 \in U_K$
- Check the reachability of (α, a_1): reachable, finish

Example

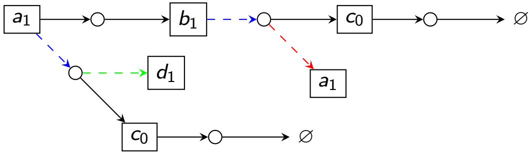
Rules: $a_1 \leftarrow b_1$, $a_1 \leftarrow d_1 \wedge c_0$, $b_1 \leftarrow c_0$, $c_1 \leftarrow b_0$
Initial state: $\alpha = \langle a_0, b_0, c_0, d_0 \rangle$
$U_K = \{(\alpha, b_1), (\alpha, d_1)\}$, $R_K = \{(\alpha, a_1)\}$



- $L = \{\{(\alpha, a_1), (\alpha, b_1), (\alpha, d_1)\}, \{(\alpha, b_1)\}, \{(\alpha, d_1)\}\}$
- Start from $\{(\alpha, b_1)\}$ and $\{(\alpha, d_1)\}$
- $b_1 \leftarrow c_0$ can be specialized to $b_1 \leftarrow c_0 \wedge a_1$ to make b_1 unreachable
- $a_1 \leftarrow d_1 \wedge c_0$ can only be generalized to $a_1 \leftarrow c_0$ as $d_1 \in U_K$
- Check the reachability of (α, a_1): reachable, finish

Example

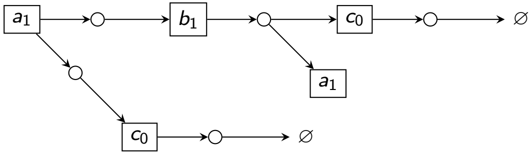
Rules: $a_1 \leftarrow b_1$, $a_1 \leftarrow d_1 \wedge c_0$, $b_1 \leftarrow c_0$, $c_1 \leftarrow b_0$
Initial state: $\alpha = \langle a_0, b_0, c_0, d_0 \rangle$
$U_K = \{(\alpha, b_1), (\alpha, d_1)\}$, $R_K = \{(\alpha, a_1)\}$



- $L = \{\{(\alpha, a_1), (\alpha, b_1), (\alpha, d_1)\}, \{(\alpha, b_1)\}, \{(\alpha, d_1)\}\}$
- Start from $\{(\alpha, b_1)\}$ and $\{(\alpha, d_1)\}$
- $b_1 \leftarrow c_0$ can be specialized to $b_1 \leftarrow c_0 \wedge a_1$ to make b_1 unreachable
- $a_1 \leftarrow d_1 \wedge c_0$ can only be generalized to $a_1 \leftarrow c_0$ as $d_1 \in U_K$
- Check the reachability of (α, a_1): reachable, finish

Example

Rules: $a_1 \leftarrow b_1$, $a_1 \leftarrow d_1 \wedge c_0$, $b_1 \leftarrow c_0$, $c_1 \leftarrow b_0$
Initial state: $\alpha = \langle a_0, b_0, c_0, d_0 \rangle$
$U_K = \{(\alpha, b_1), (\alpha, d_1)\}$, $R_K = \{(\alpha, a_1)\}$



- $L = \{\{(\alpha, a_1), (\alpha, b_1), (\alpha, d_1)\}, \{(\alpha, b_1)\}, \{(\alpha, d_1)\}\}$
- Start from $\{(\alpha, b_1)\}$ and $\{(\alpha, d_1)\}$
- $b_1 \leftarrow c_0$ can be specialized to $b_1 \leftarrow c_0 \wedge a_1$ to make b_1 unreachable
- $a_1 \leftarrow d_1 \wedge c_0$ can only be generalized to $a_1 \leftarrow c_0$ as $d_1 \in U_K$
- Check the reachability of (α, a_1): reachable, finish

Conclusion

- Given background knowledge (reachability properties), the learned models are evaluated *via* LCG
- Using classical specialization/generalization, the models learned by LF1T are revised while keeping consistent with the observation (time series data)

Ongoing work:

- Application in biological networks, *e.g.* mammalian circadian clock modeling

$\Rightarrow$ Exploit biologists knowledge to deal with few available data

References

Maxime Folschette, Loïc Paulevé, Morgan Magnin, and Olivier Roux.
Sufficient conditions for reachability in automata networks with priorities.
Theoretical Computer Science, 608:66–83, 2015.

Loïc Paulevé, Morgan Magnin, and Olivier Roux.
Static analysis of biological regulatory networks dynamics using abstract interpretation.
Mathematical Structures in Computer Science, 22(04):651–685, 2012.

Tony Ribeiro and Katsumi Inoue.
Learning prime implicant conditions from interpretation transition.
In *Inductive Logic Programming*, pages 108–125. Springer, 2015.

Thank you!