Modeling and Learning Dynamic Biological Systems with Discrete Networks

Maxime Folschette

Univ. Lille, CNRS, **Centrale Lille**, UMR 9189 **CRIStAL**, F-59000 Lille, France

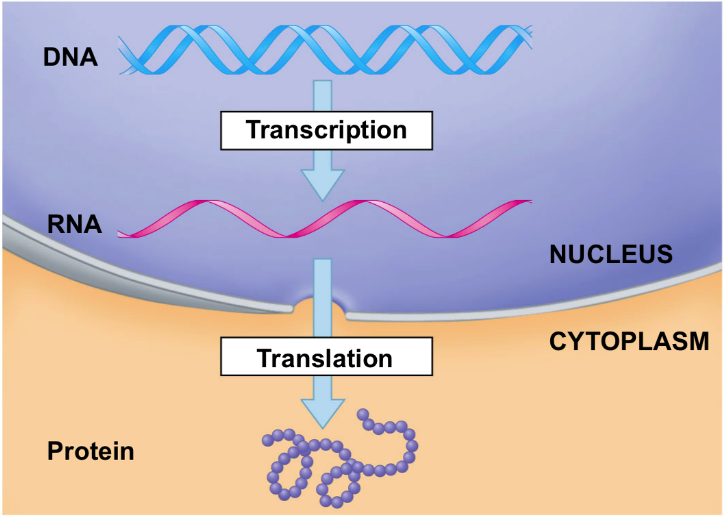
2024-03-14
Bonsai Seminar

Joint work with:
- Tony Ribeiro (Univ. Nantes & independant researcher)
- Morgan Magnin (Centrale Nantes)
- Katsumi Inoue (National Institute of Informatics, Japan)
- Omar Ikne (Univ. Lille)

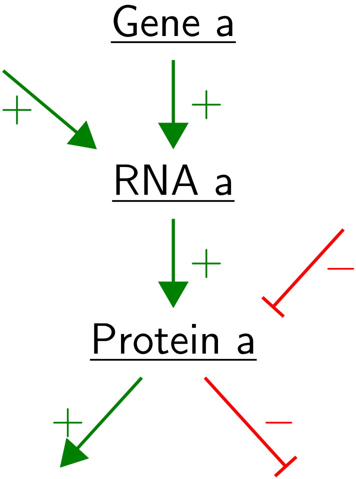
Outline

- Biological regulatory networks
- LFIT: an approach to learn a discrete model from its stage graph
- Application to phytoplankton monitoring

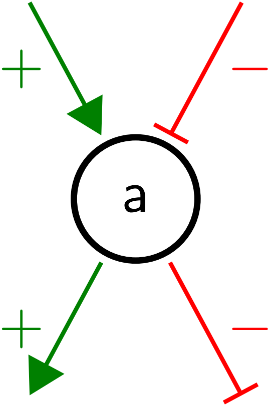
Preliminary Abstraction



Preliminary Abstraction



Preliminary Abstraction

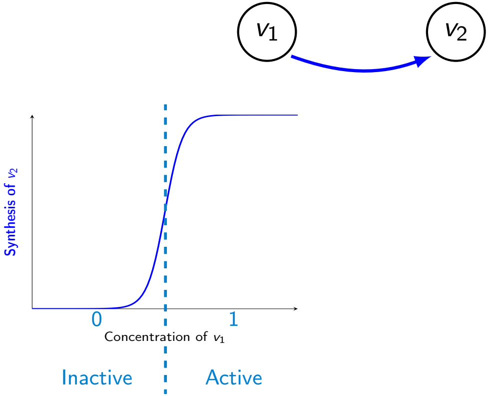


Origin of René Thomas Modeling

[Thomas, *Journal of Theoretical Biology*, 1973]

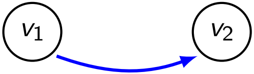
Origin of René Thomas Modeling

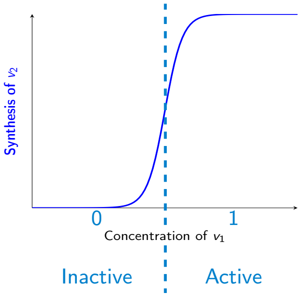
[Thomas, *Journal of Theoretical Biology*, 1973]



Origin of René Thomas Modeling

[Thomas, *Journal of Theoretical Biology*, 1973]

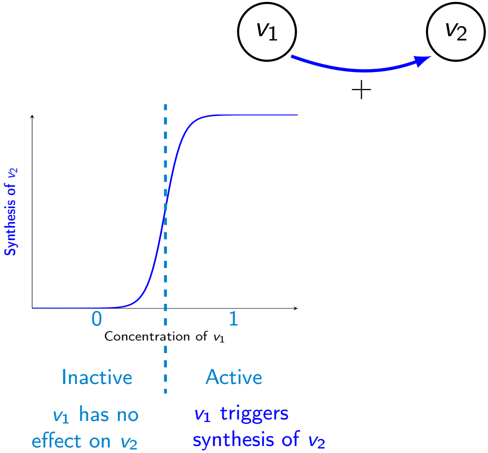




v_1 has no effect on v_2

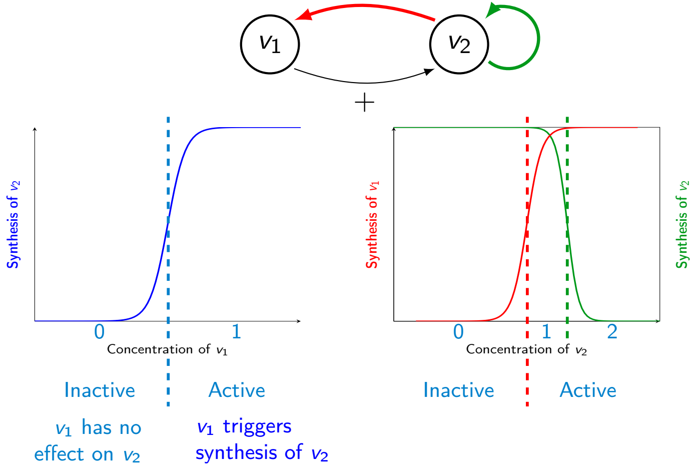
Origin of René Thomas Modeling

[Thomas, *Journal of Theoretical Biology*, 1973]



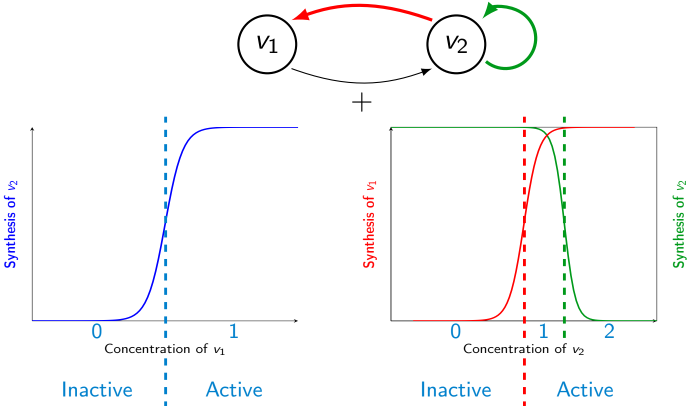
Origin of René Thomas Modeling

[Thomas, *Journal of Theoretical Biology*, 1973]



Origin of René Thomas Modeling

[Thomas, *Journal of Theoretical Biology*, 1973]



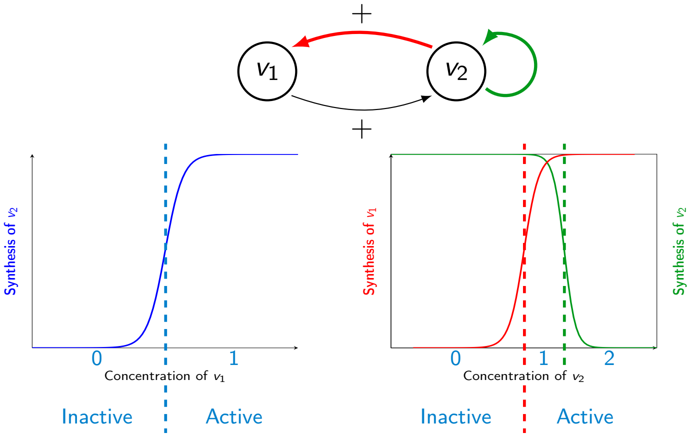
v_1 has no effect on v_2

v_1 triggers synthesis of v_2

v_2 has no effect on v_1 or itself

Origin of René Thomas Modeling

[Thomas, *Journal of Theoretical Biology*, 1973]



v_1 has no effect on v_2

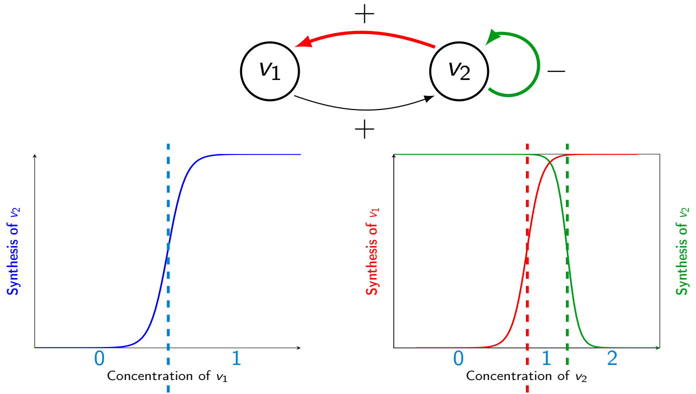
v_1 triggers synthesis of v_2

v_2 has no effect on v_1 or itself

v_2 triggers synthesis of v_1

Origin of René Thomas Modeling

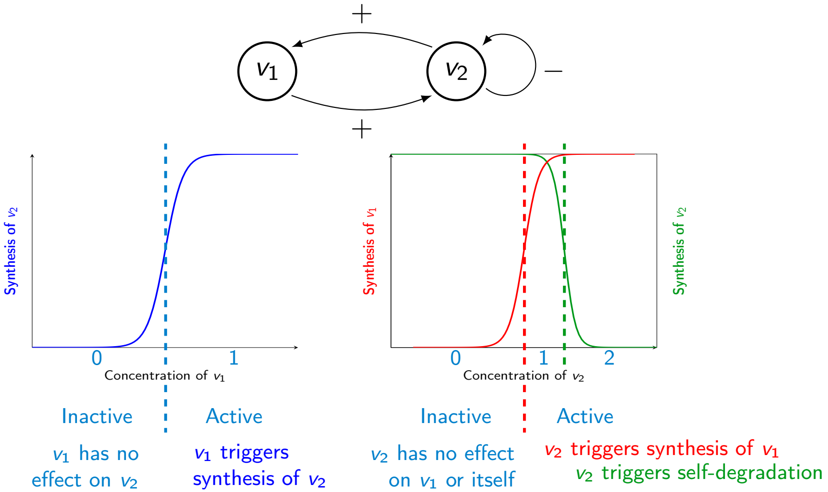
[Thomas, *Journal of Theoretical Biology*, 1973]



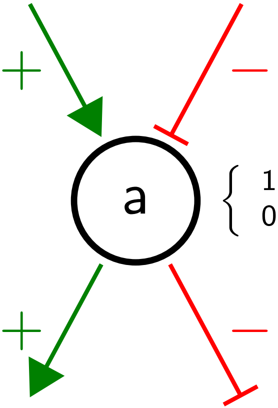
Inactive	Active	Inactive	Active
v_1 has no effect on v_2	v_1 triggers synthesis of v_2	v_2 has no effect on v_1 or itself	v_2 triggers synthesis of v_1 v_2 triggers self-degradation

Origin of René Thomas Modeling

[Thomas, *Journal of Theoretical Biology*, 1973]

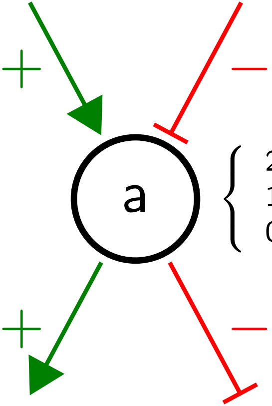


Preliminary Abstraction



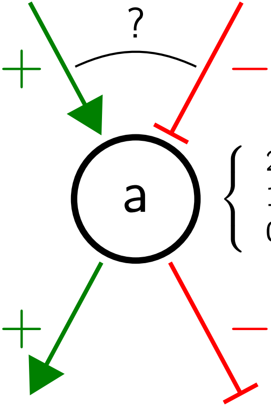
$$\left\{ \begin{array}{l} 1 = \text{active} \\ 0 = \text{inactive} \end{array} \right\}$$

Preliminary Abstraction



$$\left\{ \begin{array}{l} 2 = \text{saturation} \\ 1 = \text{traces} \\ 0 = \text{complete degradation} \end{array} \right\}$$

Preliminary Abstraction



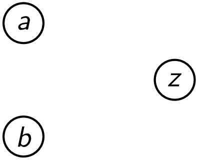
$$\left\{ \begin{array}{l} 2 = \text{saturation} \\ 1 = \text{traces} \\ 0 = \text{complete degradation} \end{array} \right\}$$

Biological Regulatory Networks

Discrete Networks / Thomas Modeling

[Kauffman, *Journal of Theoretical Biology*, 1969]
[Thomas, *Journal of Theoretical Biology*, 1973]

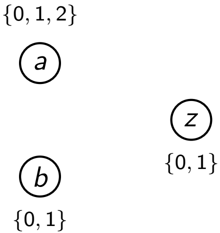
- A set of components $N = \{a, b, z\}$



Discrete Networks / Thomas Modeling

[Kauffman, *Journal of Theoretical Biology*, 1969]
[Thomas, *Journal of Theoretical Biology*, 1973]

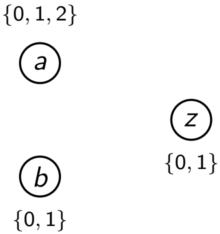
- A set of components $N = \{a, b, z\}$
- A discrete domain for each component $\text{dom}(a) = \{0, 1, 2\}$



Discrete Networks / Thomas Modeling

[Kauffman, *Journal of Theoretical Biology*, 1969]
[Thomas, *Journal of Theoretical Biology*, 1973]

- A set of components $N = \{a, b, z\}$
- A discrete domain for each component $\mathrm{dom}(a) = \{0, 1, 2\}$
- Discrete parameters / evolution functions $f_a : \mathcal{S} \to \mathrm{dom}(a)$



a	f_b
0	0
1	1
2	1

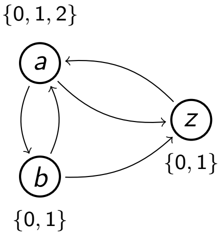
z	b	f_a
0	0	1
0	1	0
1	0	1
1	1	2

a	b	f_z
0	0	0
0	1	0
1	0	0
1	1	0
2	0	0
2	1	1

Discrete Networks / Thomas Modeling

[Kauffman, *Journal of Theoretical Biology*, 1969]
[Thomas, *Journal of Theoretical Biology*, 1973]

- A set of components $N = \{a, b, z\}$
- A discrete domain for each component $\mathrm{dom}(a) = \{0, 1, 2\}$
- Discrete parameters / evolution functions $f_a : \mathcal{S} \to \mathrm{dom}(a)$
- Signs & thresholds on the edges (redundant) $a \longrightarrow z$



a	f_b
0	0
1	1
2	1

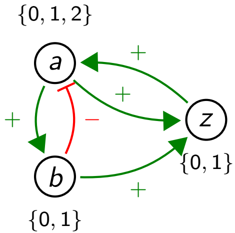
z	b	f_a
0	0	1
0	1	0
1	0	1
1	1	2

a	b	f_z
0	0	0
0	1	0
1	0	0
1	1	0
2	0	0
2	1	1

Discrete Networks / Thomas Modeling

[Kauffman, *Journal of Theoretical Biology*, 1969]
[Thomas, *Journal of Theoretical Biology*, 1973]

- A set of components $N = \{a, b, z\}$
- A discrete domain for each component $\text{dom}(a) = \{0, 1, 2\}$
- Discrete parameters / evolution functions $f_a : \mathcal{S} \to \text{dom}(a)$
- Signs & thresholds on the edges (redundant) $a \xrightarrow{+} z$



a	f_b
0	0
1	1
2	1

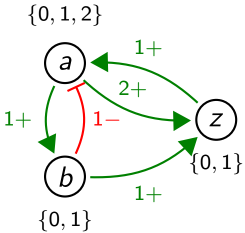
z	b	f_a
0	0	1
0	1	0
1	0	1
1	1	2

a	b	f_z
0	0	0
0	1	0
1	0	0
1	1	0
2	0	0
2	1	1

Discrete Networks / Thomas Modeling

[Kauffman, *Journal of Theoretical Biology*, 1969]
[Thomas, *Journal of Theoretical Biology*, 1973]

- A set of components $N = \{a, b, z\}$
- A discrete domain for each component $\text{dom}(a) = \{0, 1, 2\}$
- Discrete parameters / evolution functions $f_a : \mathcal{S} \to \text{dom}(a)$
- Signs & thresholds on the edges (redundant) $a \xrightarrow{2+} z$



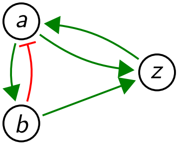
a	f_b
0	0
1	1
2	1

z	b	f_a
0	0	1
0	1	0
1	0	1
1	1	2

a	b	f_z
0	0	0
0	1	0
1	0	0
1	1	0
2	0	0
2	1	1

State Graph

The state graph depicts explicitly the dynamics



abz

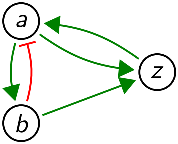
000	010	001	011
100	110	101	111
200	210	201	211

z	b	f_a
0	0	1
0	1	0
1	0	1
1	1	2

a	b	f_z
0	0	0
0	1	0
1	0	0
1	1	0
2	0	0
2	1	1

a	f_b
0	0
1	1
2	1

State Graph



The state graph depicts explicitly the dynamics

abz

000	010	001	011
100 →	110	101	111
200	210	201	211

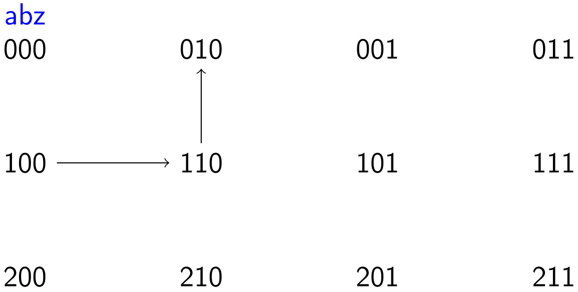
z	b	f_a
0	0	1
0	1	0
1	0	1
1	1	2

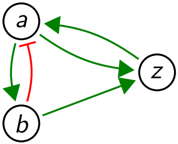
a	b	f_z
0	0	0
0	1	0
1	0	0
1	1	0
2	0	0
2	1	1

a	f_b
0	0
1	1
2	1

State Graph

The state graph depicts explicitly the dynamics





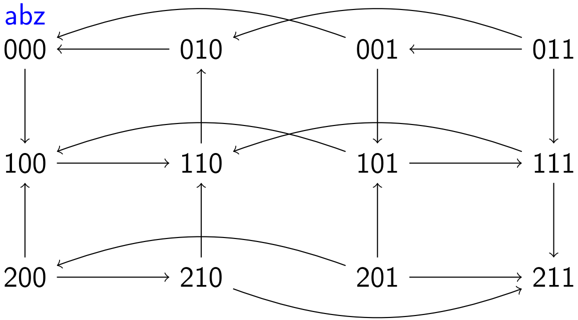
z	b	f_a
0	0	1
0	1	0
1	0	1
1	1	2

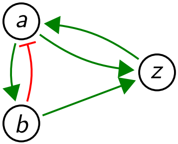
a	b	f_z
0	0	0
0	1	0
1	0	0
1	1	0
2	0	0
2	1	1

a	f_b
0	0
1	1
2	1

State Graph

The state graph depicts explicitly the dynamics





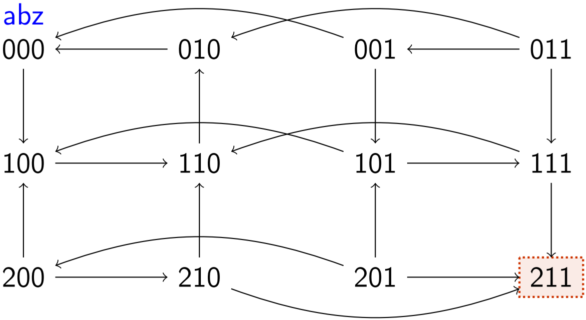
z	b	f_a
0	0	1
0	1	0
1	0	1
1	1	2

a	b	f_z
0	0	0
0	1	0
1	0	0
1	1	0
2	0	0
2	1	1

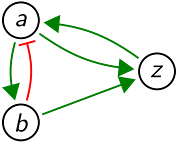
a	f_b
0	0
1	1
2	1

State Graph

The state graph depicts explicitly the dynamics



- **Stable state** = state with no successors

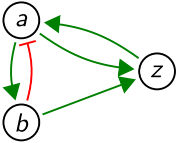


z	b	f_a
0	0	1
0	1	0
1	0	1
1	1	2

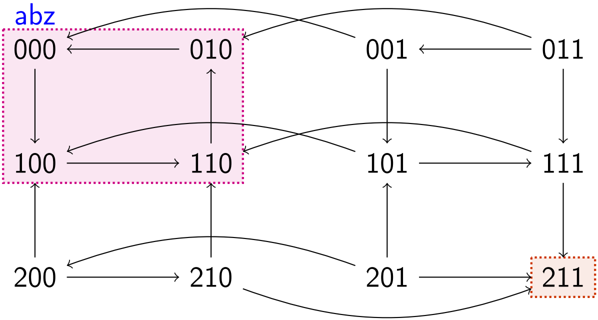
a	b	f_z
0	0	0
0	1	0
1	0	0
1	1	0
2	0	0
2	1	1

a	f_b
0	0
1	1
2	1

State Graph



The state graph depicts explicitly the dynamics



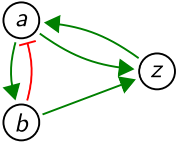
z	b	f_a
0	0	1
0	1	0
1	0	1
1	1	2

a	b	f_z
0	0	0
0	1	0
1	0	0
1	1	0
2	0	0
2	1	1

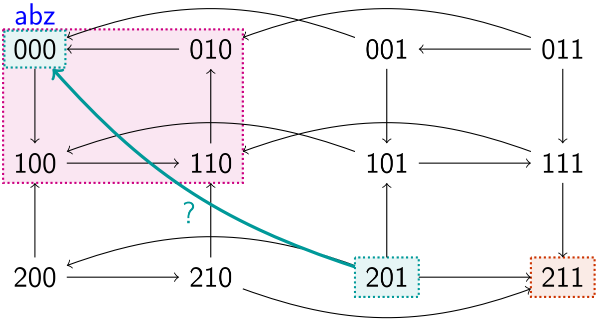
a	f_b
0	0
1	1
2	1

- **Stable state** = state with no successors
- **Complex attractor** = minimal loop or composition of loops from which the dynamics cannot escape

State Graph



The state graph depicts explicitly the dynamics



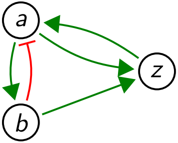
z	b	f_a
0	0	1
0	1	0
1	0	1
1	1	2

a	b	f_z
0	0	0
0	1	0
1	0	0
1	1	0
2	0	0
2	1	1

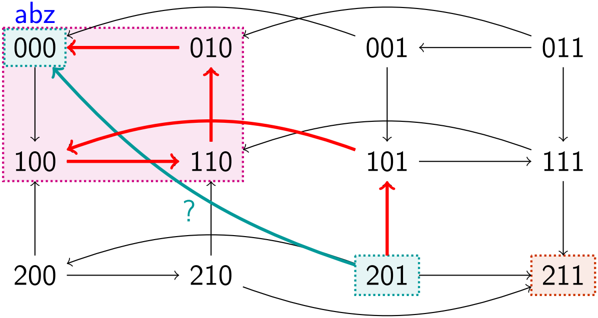
a	f_b
0	0
1	1
2	1

- **Stable state** = state with no successors
- **Complex attractor** = minimal loop or composition of loops from which the dynamics cannot escape
- **Reachability** = from 201, can I reach 000?

State Graph



The state graph depicts explicitly the dynamics



z	b	f_a
0	0	1
0	1	0
1	0	1
1	1	2

a	b	f_z
0	0	0
0	1	0
1	0	0
1	1	0
2	0	0
2	1	1

a	f_b
0	0
1	1
2	1

- **Stable state** = state with no successors
- **Complex attractor** = minimal loop or composition of loops from which the dynamics cannot escape
- **Reachability** = from 201, can I reach 000?

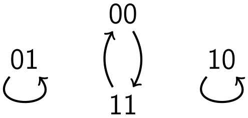
Semantics

$f_a = \neg b \qquad f_b = \neg a$

b	f_a
0	1
1	0

a	f_b
0	1
1	0

State transitions differ according to the update semantics used:



Synchronous

- **Synchronous**: all variables are updated

Semantics

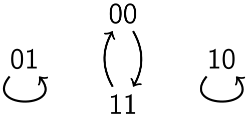
$f_a = \neg b$

b	f_a
0	1
1	0

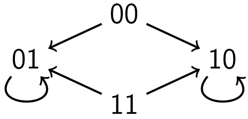
$f_b = \neg a$

a	f_b
0	1
1	0

State transitions differ according to the update semantics used:



Synchronous



Asynchronous

- **Synchronous**: all variables are updated
- **Asynchronous**: only one variable is updated

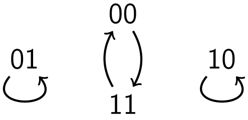
Semantics

$f_a = \neg b$ $\quad$ $f_b = \neg a$

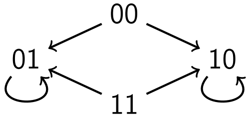
b	f_a
0	1
1	0

a	f_b
0	1
1	0

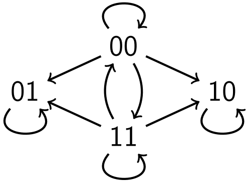
State transitions differ according to the update semantics used:



Synchronous



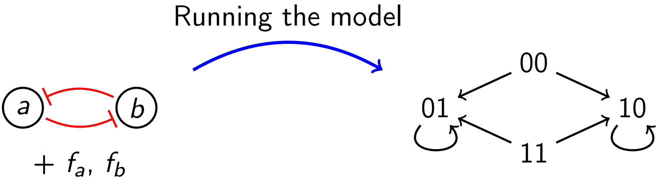
Asynchronous



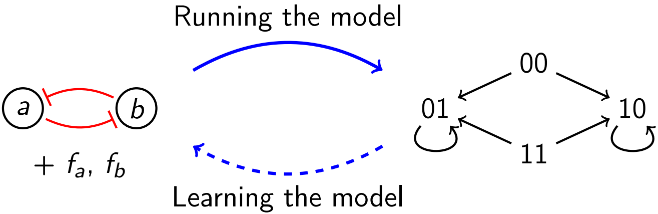
General

- **Synchronous**: all variables are updated
- **Asynchronous**: only one variable is updated
- **General**: any number of variables can be updated

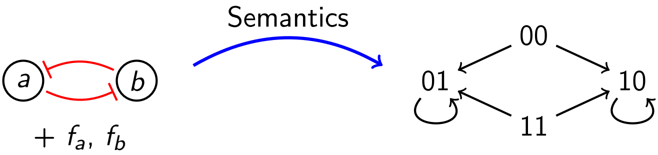
Learning from the State Graph



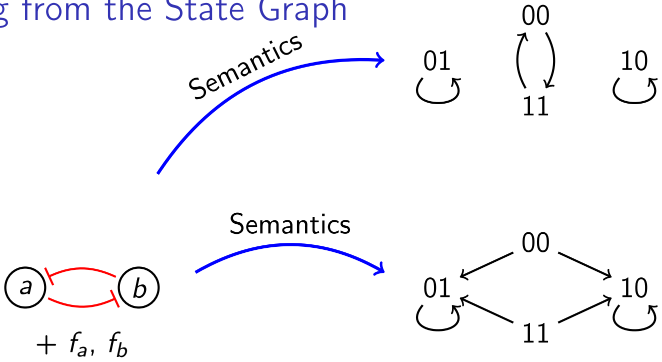
Learning from the State Graph



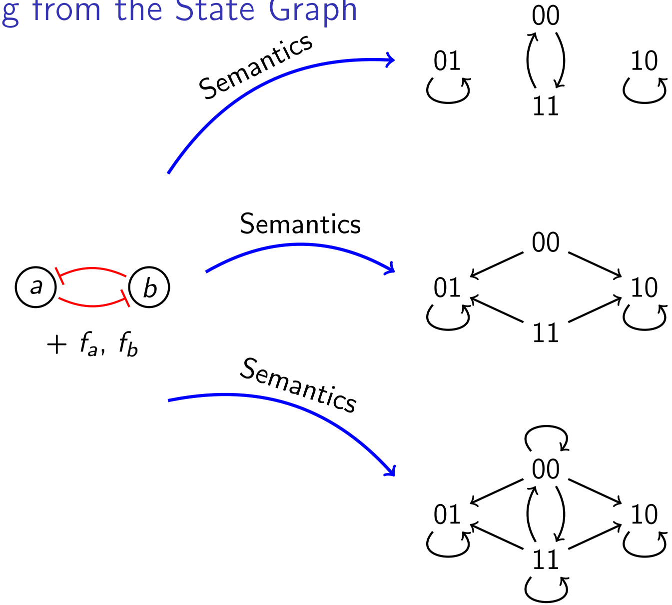
Learning from the State Graph



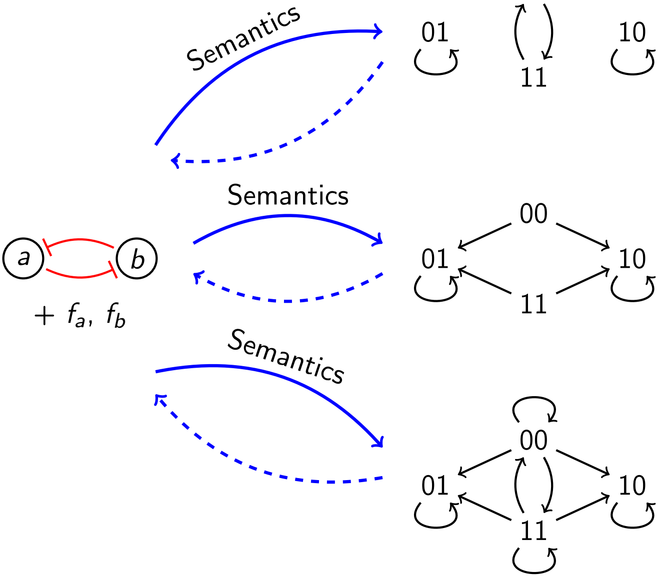
Learning from the State Graph



Learning from the State Graph



Learning from the State Graph



Logic Rules

A logic program is a set of logic rules.
It is an alternative representation of biological networks.

$$a_1 \leftarrow a_0, b_0, c_2.$$

If a and b are at level 0 and c is at level 2, then a can change its value to 1.

$$a_1 \leftarrow c_2.$$

Whenever c is at level 2, a can change its value to 1.

$$a_1 \leftarrow .$$

a can change its value to 1 anytime.

Logic Rules

A logic program is a set of logic rules.
It is an alternative representation of biological networks.

$$a_1 \leftarrow a_0, b_0, c_2.$$

If a and b are at level 0 and c is at level 2, then a can change its value to 1.

$$a_1 \leftarrow c_2.$$

Whenever c is at level 2, a can change its value to 1.

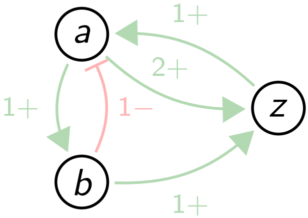
$$a_1 \leftarrow .$$

a can change its value to 1 anytime.

The same notion of **semantics** applies.

Discrete Models as Logic Programs

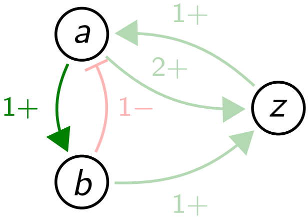
Discrete model:



Logic program:

Discrete Models as Logic Programs

Discrete model:



Logic program:

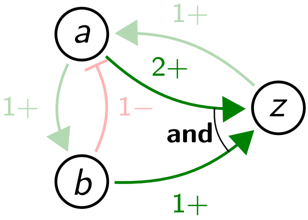
$b_1 \leftarrow a_1.$

$b_1 \leftarrow a_2.$

$b_0 \leftarrow a_0.$

Discrete Models as Logic Programs

Discrete model:



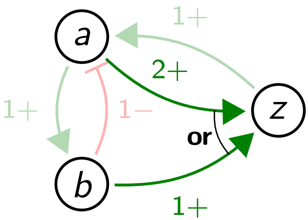
Logic program:

$b_1 \leftarrow a_1.$
$b_1 \leftarrow a_2.$
$b_0 \leftarrow a_0.$

$z_1 \leftarrow a_2 \wedge b_1.$
$z_0 \leftarrow a_0.$
$z_0 \leftarrow a_1.$
$z_0 \leftarrow b_0.$

Discrete Models as Logic Programs

Discrete model:



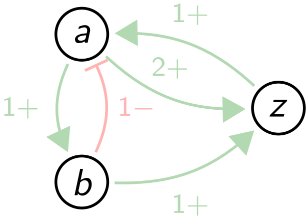
Logic program:

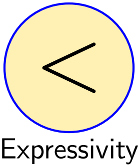
$b_1 \leftarrow a_1.$
$b_1 \leftarrow a_2.$
$b_0 \leftarrow a_0.$

$z_1 \leftarrow a_2.$
$z_1 \leftarrow b_1.$
$z_0 \leftarrow a_1 \wedge b_0.$
$z_0 \leftarrow a_0 \wedge b_0.$

Discrete Models as Logic Programs

Discrete model:





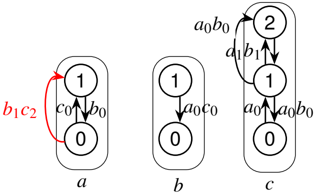
Logic program:

$b_1 \leftarrow a_1.$
$b_1 \leftarrow a_2.$
$b_0 \leftarrow a_0.$

$z_1 \leftarrow a_2.$
$z_1 \leftarrow b_1.$
$z_0 \leftarrow a_1 \wedge b_0.$
$z_0 \leftarrow a_0 \wedge b_0.$

AANs as Logic Programs

Asynchronous automata network:



Picture: [Soh et al., CMSB'2023]

Logic program:

$b_1 \leftarrow a_1.$

$b_1 \leftarrow a_2.$

$b_0 \leftarrow a_0.$

$z_1 \leftarrow a_2.$

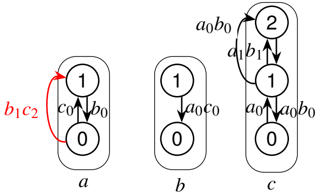
$z_1 \leftarrow b_1.$

$z_0 \leftarrow a_1 \wedge b_0.$

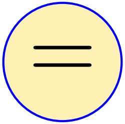
$z_0 \leftarrow a_0 \wedge b_0.$

AANs as Logic Programs

Asynchronous automata network:



Picture: [Soh et al., CMSB'2023]



Expressivity

Logic program:

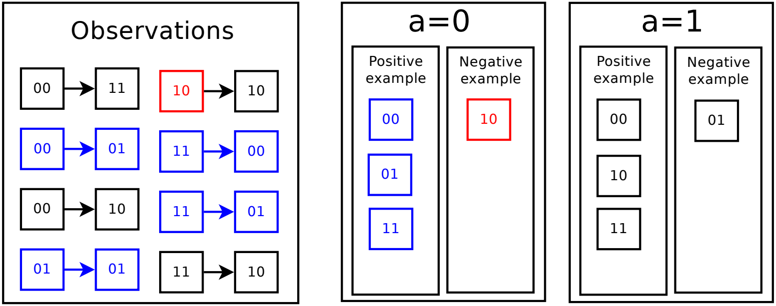
$b_1 \leftarrow a_1.$
$b_1 \leftarrow a_2.$
$b_0 \leftarrow a_0.$

$z_1 \leftarrow a_2.$
$z_1 \leftarrow b_1.$
$z_0 \leftarrow a_1 \wedge b_0.$
$z_0 \leftarrow a_0 \wedge b_0.$

Learning From Interpretation Transition (LFIT)

Learning Algorithm Intuition: Classification Problem

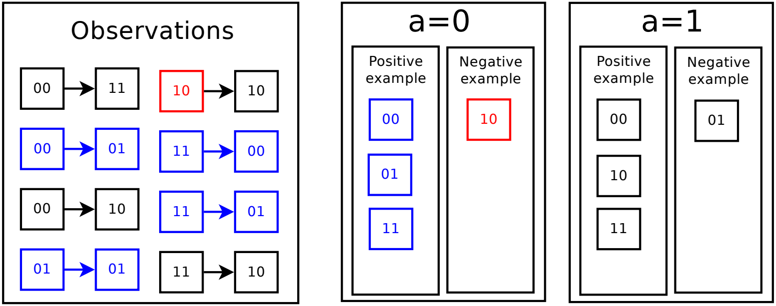
Learn applicable rules: conditions so that a variable **can** take a certain value in next state.



Equivalent to a **classification problem**: What is a typical state where a can take value 0 in the next state ? Here: when a_0 or b_1 is present.

Learning Algorithm Intuition: Classification Problem

Learn applicable rules: conditions so that a variable **can** take a certain value in next state.



Equivalent to a **classification problem**: What is a typical state where a can take value 0 in the next state ? Here: when a_0 or b_1 is present.

That is: $a_0 \leftarrow a_0.$ $a_0 \leftarrow b_1.$

Presentation of GULA

GULA = General Usage LFIT Algorithm

Input: a set of transitions ($s_1 \rightarrow s_2$)

Output: a logic program that reproduces the input

Principle: minimal refinements of the rules

Compatible with the synchronous, asynchronous and general semantics (and any semantics without memory or "hard-coded" behaviors)

GULA: Initial Logic Program

Suppose:

- a and b have two levels $\{0, 1\}$ and c has three levels $\{0, 1, 2\}$

GULA starts with the most general program:

$$a_0 \leftarrow . \qquad b_0 \leftarrow . \qquad c_0 \leftarrow .$$
$$a_1 \leftarrow . \qquad b_1 \leftarrow . \qquad c_1 \leftarrow .$$
$$c_2 \leftarrow .$$

With this program, everything is always possible

GULA: One Step of Minimal Refinements

Suppose:

- a and b have two levels $\{0, 1\}$ and c has three levels $\{0, 1, 2\}$
- the current program contains the following rules regarding a_1:

$$a_1 \leftarrow c_2. \qquad\qquad a_1 \leftarrow b_1.$$

- from state $\langle a_1, b_0, c_2 \rangle$, a_1 is never observed in the next states.

However, the first rule allows this; it is then necessary to make **minimal refinements** in order to make this rule inapplicable:

GULA: One Step of Minimal Refinements

Suppose:

- a and b have two levels $\{0, 1\}$ and c has three levels $\{0, 1, 2\}$
- the current program contains the following rules regarding a_1:

$$a_1 \leftarrow c_2. \qquad\qquad a_1 \leftarrow b_1.$$

- from state $\langle a_1, b_0, c_2 \rangle$, a_1 is never observed in the next states.

However, the first rule allows this; it is then necessary to make **minimal refinements** in order to make this rule inapplicable:

$$a_1 \leftarrow a_0, c_2.$$
$$a_1 \leftarrow b_1, c_2.$$
$$a_1 \leftarrow c_2, c_0.$$
$$a_1 \leftarrow c_2, c_1.$$

$$a_1 \leftarrow b_1.$$
(No change)

GULA: One Step of Minimal Refinements

Suppose:

- a and b have two levels $\{0, 1\}$ and c has three levels $\{0, 1, 2\}$
- the current program contains the following rules regarding a_1:

$$a_1 \leftarrow c_2. \qquad\qquad a_1 \leftarrow b_1.$$

- from state $\langle a_1, b_0, c_2 \rangle$, a_1 is never observed in the next states.

However, the first rule allows this; it is then necessary to make **minimal refinements** in order to make this rule inapplicable:

$$a_1 \leftarrow a_0, c_2. \qquad\qquad a_1 \leftarrow b_1.$$

$$a_1 \leftarrow b_1, c_2.$$

$$a_1 \leftarrow c_2, c_0.$$

$$a_1 \leftarrow c_2, c_1.$$

GULA: One Step of Minimal Refinements

Suppose:

- a and b have two levels $\{0, 1\}$ and c has three levels $\{0, 1, 2\}$
- the current program contains the following rules regarding a_1:

$$a_1 \leftarrow c_2. \qquad\qquad a_1 \leftarrow b_1.$$

- from state $\langle a_1, b_0, c_2 \rangle$, a_1 is never observed in the next states.

However, the first rule allows this; it is then necessary to make **minimal refinements** in order to make this rule inapplicable:

$$a_1 \leftarrow a_0, c_2.$$
$$a_1 \leftarrow b_1, c_2.$$

$$a_1 \leftarrow b_1.$$

(More general)

GULA: One Step of Minimal Refinements

Suppose:

- a and b have two levels $\{0, 1\}$ and c has three levels $\{0, 1, 2\}$
- the current program contains the following rules regarding a_1:

$$a_1 \leftarrow c_2. \qquad\qquad a_1 \leftarrow b_1.$$

- from state $\langle a_1, b_0, c_2 \rangle$, a_1 is never observed in the next states.

However, the first rule allows this; it is then necessary to make **minimal refinements** in order to make this rule inapplicable:

$$a_1 \leftarrow a_0, c_2. \qquad\qquad a_1 \leftarrow b_1.$$

GULA: Final Result

The output of GULA respects some good properties:

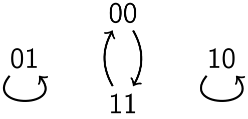
- **Consistency**: the program allows no negative examples
- **Realization**: the program covers all positive examples
- **Completeness**: the program covers all the state space
- **Minimality** of the rules (most general conditions)

Example: Synchronous Semantics

$f_a = \neg b$ $\qquad$ $f_b = \neg a$

b	f_a
0	1
1	0

a	f_b
0	1
1	0



Synchronous

$// \ f_a = \neg b$

$a_0 \leftarrow b_1$

$a_1 \leftarrow b_0$

$// \ f_b := \neg a$

$b_0 \leftarrow a_1$

$b_1 \leftarrow a_0$

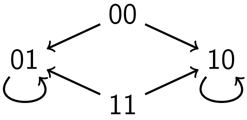
Example: Asynchronous Semantics

$f_a = \neg b$

b	f_a
0	1
1	0

$f_b = \neg a$

a	f_b
0	1
1	0



Asynchronous

// $f_a = \neg b$

$a_0 \leftarrow b_1$

$a_1 \leftarrow b_0$

// $f_b = \neg a$

$b_0 \leftarrow a_1$

$b_1 \leftarrow a_0$

// Default rules

$a_0 \leftarrow a_0$

$a_1 \leftarrow a_1$

$b_0 \leftarrow b_0$

$b_1 \leftarrow b_1$

Results

GULA: an algorithm to learn a biological regulatory network

- From the state graph
- In order to recover the structure of the model
- Applicable to a widespread class of semantics

Limitations:

- Exponential complexity
  - **PRIDE**: a greedy polynomial version of **GULA**
- What if the data is **incomplete** or **noisy**?
  - Heuristic to avoid overfitting

Heuristic: Weighted Likeliness/Unlikeliness Rules

- Use the algorithm twice to learn two logic programs:
  - likeliness rules: what is possible
  - unlikeliness rules: what is impossible
- Weight each rule by the number of observations it matches

Likeliness rules	**Unlikeliness rules**
$(3, a_0 \leftarrow b_1)$	$(30, a_0 \leftarrow c_1)$
$(15, a_1 \leftarrow b_0)$	$(5, a_1 \leftarrow c_0)$
$\vdots$	$\vdots$

Heuristic: Using Weighted Likeliness/Unlikeliness Rules

Explainable predictions:

- Compare weights of applicable likeliness/unlikeliness rules
- Ratio of highest weights ⇒ **probability** P
- Rules with highest weights ⇒ **explanation** E

$$\text{predict} : (atom, state) \mapsto (P, E)$$

Likeliness rules	**Unlikeliness rules**
$(3, a_0 \leftarrow b_1)$	$(30, a_0 \leftarrow c_1)$
$(15, a_1 \leftarrow b_0)$	$(5, a_1 \leftarrow c_0)$

Heuristic: Using Weighted Likeliness/Unlikeliness Rules

Explainable predictions:

- Compare weights of applicable likeliness/unlikeliness rules
- Ratio of highest weights ⇒ **probability** P
- Rules with highest weights ⇒ **explanation** E

$$\text{predict} : (atom, state) \mapsto (P, E)$$

Likeliness rules
$(3, a_0 \leftarrow b_1)$
$(15, a_1 \leftarrow b_0)$

Unlikeliness rules
$(30, a_0 \leftarrow c_1)$
$(5, a_1 \leftarrow c_0)$

$\text{predict}(a_1, \langle a_1, b_0, c_0 \rangle) = (0.75, ((15, a_1 \leftarrow b_0), (5, a_1 \leftarrow c_0)))$ ⇒ Likely

Heuristic: Using Weighted Likeliness/Unlikeliness Rules

Explainable predictions:

- Compare weights of applicable likeliness/unlikeliness rules
- Ratio of highest weights ⇒ **probability** P
- Rules with highest weights ⇒ **explanation** E

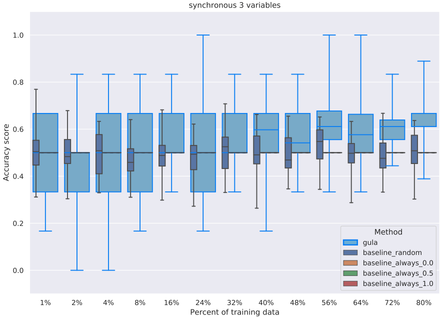
$$\text{predict} : (atom, state) \mapsto (P, E)$$

Likeliness rules	**Unlikeliness rules**
$(3, a_0 \leftarrow b_1)$	$(30, a_0 \leftarrow c_1)$
$(15, a_1 \leftarrow b_0)$	$(5, a_1 \leftarrow c_0)$

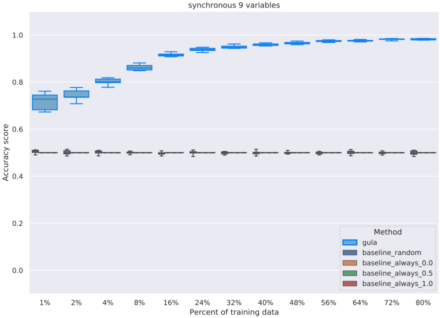
$\text{predict}(a_1, \langle a_1, b_0, c_0 \rangle) = (0.75, ((15, a_1 \leftarrow b_0), (5, a_1 \leftarrow c_0)))$ ⇒ Likely

$\text{predict}(a_0, \langle a_1, b_1, c_1 \rangle) = (0.09, ((3, a_0 \leftarrow b_1), (30, a_0 \leftarrow c_1)))$ ⇒ Unlikely

Prediction power



3 variables



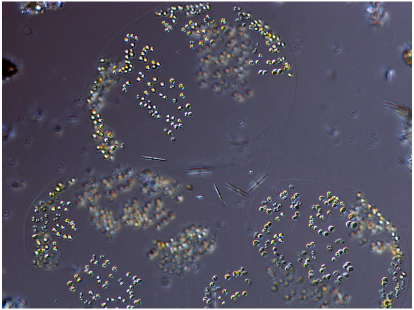
9 variables

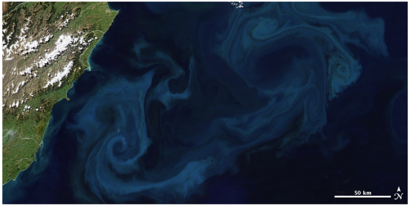
Training data = X% of transitions

Tested against unseen states (not in the training data)

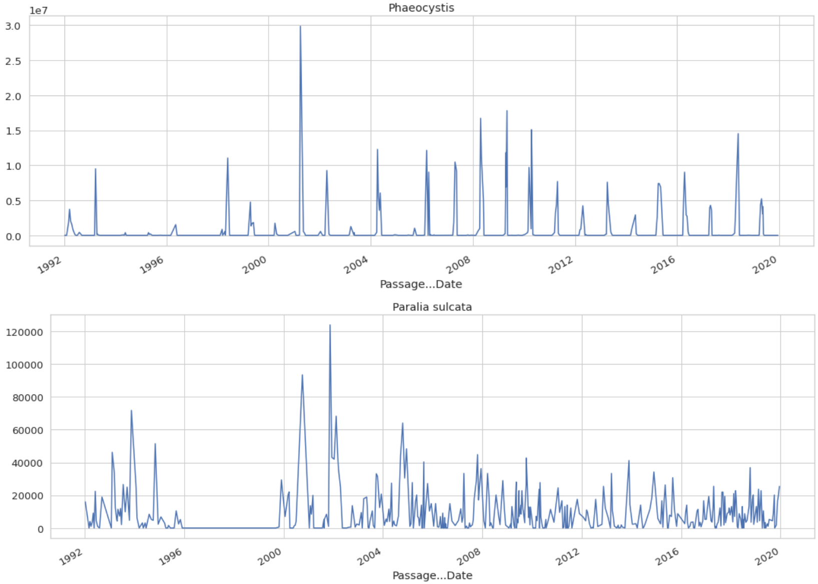
Application: Dynamics of Marine Phytoplankton

Phytoplankton Blooms

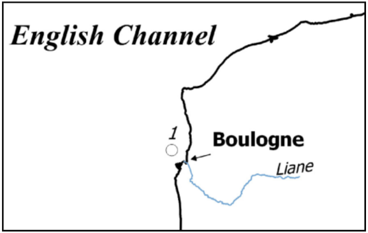




Phytoplankton Blooms



SRN Dataset



https://www.seanoe.org/data/00397/50832/

Sampling location	Sampling date	Taxon	Value	Sampling depth
001-P-015	1992-05-18	CHLOROA	6.0	Surface (0-1m)
006-P-001	2019-12-02	Chaetoceros	1000.0	Surface (0-1m)
002-P-007	1994-05-25	Pleurosigma	100.0	Surface (0-1m)
002-P-030	2005-10-19	SALI	34.83	Surface (0-1m)
006-P-007	2015-09-28	Guinardia delicatula	11400.0	Surface (0-1m)

Environmental variables (7)   Phytoplankton species (12)

Applying LFIT

Expectations

- Find known **abiotic** influences (of environment on phytoplankton)
- Find new **biotic** influences (of phytoplankton species on others)

Input

- Pre-processing: data cleaning + discretization
- Train set: 253 transitions
- Test set: 53 transitions

Output

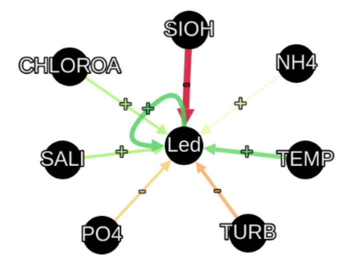
- Run time = 2.35s (**PRIDE**, greedy version of **GULA**)
- **1683** likeliness rules & **1981** unlikeliness rules
- Model accuracy: **0.670**

Global Influences

Process: Search and count patterns in rules that characterize an activation/inhibition
Result: Score $[-1; +1]$ between each pair of variables

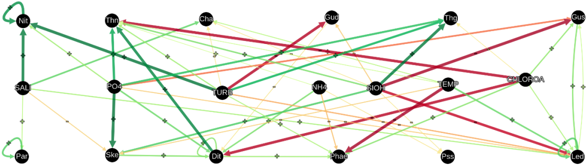
Influences on phytoplankton species Led:

Variable	Positive	Negative	Global
PO4	$+0$	-58	-0.36
SALI	$+71$	-4	$+0.42$
CHLOROA	$+84$	-22	$+0.39$
SIOH	$+3$	-161	-0.98
NH4	$+25$	-5	$+0.12$
TEMP	$+106$	-5	$+0.63$
TURB	$+10$	-87	-0.48

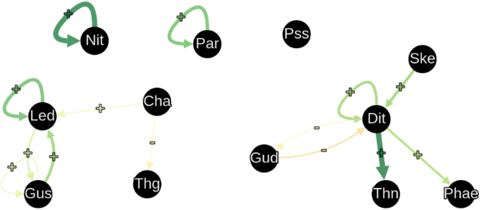


$$\text{global\_influence}(\texttt{PO4} \rightarrow \texttt{Led}) = \frac{+0 + (-58)}{161} = -0.36$$

Results



Global influence graph (biotic and abiotic interactions)



Biotic interactions (between phytoplankton only)

Very few biotic interactions...
Ongoing work: integrate knowledge + validate results

Conclusion

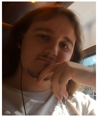
Conclusion

- **Learn** biological regulatory networks with LFIT
- **Heuristics** to tackle real data
  - Good results with 10% of the transitions
- Ongoing: **Application** to phytoplankton
- You can try **GULA** at home:
  `https://github.com/Tony-sama/pylfit`

Outlooks:

- **PRIDE**: polynomial algorithm that "misses" some explanations
- Improve the application (integrate existing knowledge)
- Improve the biological network inference

Thanks



Tony
RIBEIRO

Omar
IKNE

Morgan
MAGNIN

Katsumi
INOUE

Cédric
LHOUSSAINE

Sébastien
LEFEBVRE

Madeleine
EYRAUD

Bibliography

- About GULA: Tony Ribeiro, Maxime Folschette, Morgan Magnin and Katsumi Inoue. **Learning any memory-less discrete semantics for dynamical systems represented by logic programs**. *Machine Learning* 111, Springer. November 2021. https://doi.org/10.1007/s10994-021-06105-4
- pyLFIT Python library: https://github.com/Tony-sama/pylfit
- About PRIDE: Tony Ribeiro, Maxime Folschette, Morgan Magnin and Katsumi Inoue. **Polynomial Algorithm For Learning From Interpretation Transition**. Poster at the *1st International Joint Conference on Learning & Reasoning*. October 2021, Online. https://hal.science/hal-03347026v1
- Application to phytoplankton: Omar Iken, Maxime Folschette and Tony Ribeiro. **Automatic Modeling of Dynamical Interactions Within Marine Ecosystems**. Poster in the *1st International Joint Conference on Learning & Reasoning*. October 2021, Online. https://hal.science/hal-03347033v1