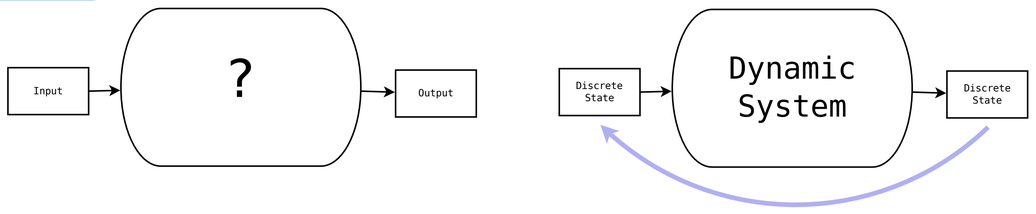
Polynomial Algorithm For Learning From Interpretation Transition

Tony Ribeiro[1,3], **Maxime Folschette**[2], **Morgan Magnin**[1,3], **Katsumi Inoue**[3]
(1) Université de Nantes, Centrale Nantes, CNRS, LS2N, F-44000 Nantes, France
(2) Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
(3) National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
tony.ribeiro@ls2n.fr, maxime.folschette@irisa.fr, morgan.magnin@ls2n.fr, oliver.roux@ls2n.fr, inoue@nii.ac.jp

Motivations: Learning Dynamics

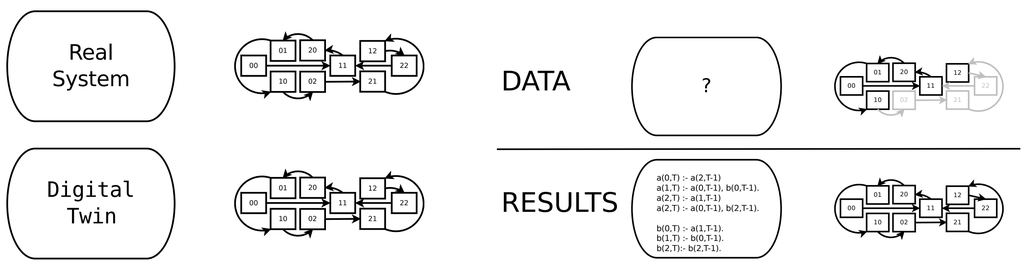
- Given a set of input/output states of a black-box system, learn its internal mechanics.
- **Discrete system:** input/output are vectors of same size which contain discrete values.
- **Dynamic system:** input/output are states of the system and output is the next input.

Problem



- **Goal:** produce an artificial system with the same behavior, i.e., a digital twin.
- **Representation:** propositional logic programs encoding multi-valued discrete variables.
- **Method:** learn the dynamics of systems from its state transitions.

Goal



Algorithm: PRIDE

Theorem 1 (Consistent Rule Always exists). *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, $(s, s') \in T$ and $v^{val} \in s'$. The rule $R = v^{val} \leftarrow s$ is consistent with T and realizes (s, s').*

Theorem 2 (Irreducible Rules are Optimal). *Let R be a rule consistent with a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. If $\forall R' \in \{\text{head}(R) \leftarrow \text{body}(R) \setminus \{v^{val}\} \mid v^{val} \in \text{body}(R)\}$, R' conflicts with T, then $\nexists R'' \neq R$ consistent with T such that $R'' \geq R$ and thus, $R \in P_{\mathcal{O}}(T)$.*

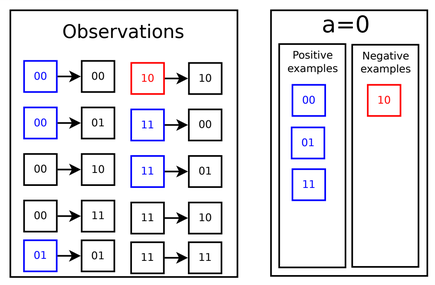
Idea:

- given positives/negatives examples of occurrence of a target atom v^{val} in T;
- we can find a rule $R \in P_{\mathcal{O}}(T)$ to explain each positive example s
- starting from $v^{val} \leftarrow s$ remove body atom until conflict is not avoidable

Algorithmic properties:

- it is faster to start from $v^{val} \leftarrow \emptyset$, specialise it until consistency and then generalise
- adding only the atoms of s ensure to matches s, in worst case we reach $v^{val} \leftarrow s$
- more variable in the system, more generalization is avoided

PRIDE



Algorithm 1 PRIDE($\mathcal{A}$, T, $\mathcal{F}$, $\mathcal{T}$)

INPUT : A set of atoms $\mathcal{A}$, a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, two sets of variables $\mathcal{F}$ and $\mathcal{T}$.
OUTPUT: A $\mathcal{DMVLP}$ $P \subseteq P_{\mathcal{O}}(T)$ s.t. P realizes T

for each $v^{val} \in \mathcal{A}$ such that $v \in \mathcal{T}$ **do**
  // 1) Extract positives and negatives examples
  $Pos_{v^{val}} := \{s \in \mathcal{S}^{\mathcal{F}} \mid \exists (s, s') \in T, v^{val} \in s'\}$
  $Neg_{v^{val}} := \{s \in \mathcal{S}^{\mathcal{F}} \mid \nexists (s, s') \in T, v^{val} \in s'\}$
  // 2) Generate the rules of v^{val} that are in $P_{\mathcal{O}}(T)$ and that match each state of $Pos_{v^{val}}$
  while $Pos_{v^{val}} \neq \emptyset$ **do**
    pick $pos \in Pos_{v^{val}}$
    $R := search(v^{val}, pos, Neg_{v^{val}})$
    // Clean other positives examples covered
    $Pos_{v^{val}} := Pos_{v^{val}} \setminus \{s \mid s \in Pos_{v^{val}}, R \sqcap s\}$
    $P = P \cup \{R\}$
return P

Algorithm 2 search(v^{val}, pos, $Neg_{v^{val}}$)

INPUT : An atom $v^{val} \in \mathcal{A}$, a state $pos \in \mathcal{S}^{\mathcal{F}}$ and a set of states $Neg_{v^{val}} \subset \mathcal{S}^{\mathcal{F}}$.
OUTPUT: A $\mathcal{MVL}$ rule $R \in P_{\mathcal{O}}(T)$ s.t. $R \sqcap pos$

$R := v^{val} \leftarrow \emptyset$
// Specialize R until consistency with $Neg_{v^{val}}$
for each $neg \in Neg_{v^{val}}$ **do**
  if $R \sqcap neg$ **then**
    pick $c \in (pos \setminus neg)$
    $R := \text{head}(R) \leftarrow \text{body}(R) \cup \{c\}$
// Generalize R while keeping consistency
for each $c \in \text{body}(R)$ **do** // Test each condition
  $R' := \text{head}(R) \leftarrow \text{body}(R) \setminus \{c\}$
  $conflict := false$
  for each $neg \in Neg_{v^{val}}$ **do**
    if $R' \sqcap neg$ **then** // Necessary condition
      $conflict := true$; **BREAK**
  if $conflict == false$ **then** // R' is valid
    $R := R'$
return R

We extract positive and negatives examples (feature states) of each target atom occurrence. Rules should match a positive and no negative while being irreducible.

Formalization: $\mathcal{MVL}$ and $\mathcal{DMVLP}$

Definition 1 (Atoms). *Let $\mathcal{V} = \{v_1, \ldots, v_n\}$ be a finite set of $n \in \mathbb{N}$ variables, and $\text{dom} : \mathcal{V} \to \mathbb{N}$. The atoms of $\mathcal{MVL}$ (denoted $\mathcal{A}$) are of the form v^{val} where $v \in \mathcal{V}$ and $val \in [\![0; \text{dom}(v)]\!]$.*

Definition 2 (Multi-valued logic program). *A $\mathcal{MVLP}$ is a set of $\mathcal{MVL}$ rules:*

$$\underbrace{v_0^{val_0}}_{head} \leftarrow \underbrace{v_1^{val_1} \wedge v_2^{val_2} \wedge v_3^{val_3} \wedge \cdots \wedge v_m^{val_m}}_{body}$$

Definition 3 (Dynamic $\mathcal{MVLP}$). *Let $\mathcal{T} \subset \mathcal{V}$ and $\mathcal{F} \subset \mathcal{V}$ such that $\mathcal{F} = \mathcal{V} \setminus \mathcal{T}$. A $\mathcal{DMVLP}$ P is a $\mathcal{MVLP}$ such that $\forall R \in P, \text{var}(\text{head}(R)) \in \mathcal{T}$ and $\forall v^{val} \in \text{body}(R), v \in \mathcal{F}$.*

Definition 4 (Discrete state). *A discrete state s on $\mathcal{T}$ (resp. $\mathcal{F}$) of a $\mathcal{DMVLP}$ is a function from $\mathcal{T}$ (resp. $\mathcal{F}$) to $\mathbb{N}$. $\mathcal{S}^{\mathcal{T}}$ (resp. $\mathcal{S}^{\mathcal{F}}$) denote the set of all discrete states of $\mathcal{T}$ (resp. $\mathcal{F}$).*

Definition 5 (Transition). *A transition is a couple of states $(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$.*

Definition 6 (Semantics). *A dynamical semantics is a function of $\left(\mathcal{DMVLP} \to (\mathcal{S}^{\mathcal{F}} \to \wp(\mathcal{S}^{\mathcal{T}}) \setminus \{\emptyset\})\right)$ where $\mathcal{DMVLP}$ is the set of $\mathcal{DMVLP}$s ($\wp$ is the power set symbol).*

- R_1 dominates R_2, written $R_1 \geq R_2$ if $head(R_1) = head(R_2)$ and $body(_1) \subseteq body(_2)$.
- R matches $s \in \mathcal{S}^{\mathcal{F}}$, written $R \sqcap s$, if $body() \subseteq s$.
- R realizes the transition $(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, if $R \sqcap s, head(R) \in s'$.
- R conflicts with $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ when $\exists (s, s') \in T, \left(R \sqcap s \wedge \forall (s, s'') \in T, head(R) \notin s''\right)$.

Definition 7 (Suitable program). *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. A $\mathcal{DMVLP}$ P is suitable for T when: P is complete, consistent with T, realizes T and $\forall R$ not conflicting with T, $\exists R' \in P$ s.t. $R \geq R'$. If in addition, $\forall R \in P$, all the rules R' belonging to a $\mathcal{MVLP}$ suitable for T are such that $R \geq R'$ implies $R' \geq R$ then P is unique, called optimal and denoted $P_{\mathcal{O}}(T)$.*

Problem: Combinatorial Explosion

GULA and **PRIDE**:

- In [2] we proposed an Algorithm (**GULA**) to learn $P_{\mathcal{O}}(T)$ but with exponential complexity.
- We introduce an heuristics algorithm **PRIDE** which trades the completeness of **GULA** for a polynomial complexity. **PRIDE** learns a subset of $P_{\mathcal{O}}(T)$ sufficient to realise T.

Run Time

System variables (n)	7	9	10	12	13	15	18	23
GULA run time	0.027s	0.157s	0.49s	2.62s	5.63s	T.O.	T.O.	T.O.
PRIDE run time	0.005s	0.02s	0.06s	0.37s	0.484s	1.55s	6.39s	32.43s

Average run time of **GULA** and **PRIDE** when learning Boolean networks of PyBoolNet [3] from at most 10,000 over 3 runs with a time-out (T.O.) of 1,000 seconds.

PRIDE performances allows to learn more complex systems and drastically reduce computation time of smaller ones.

Implementation: python library and user API

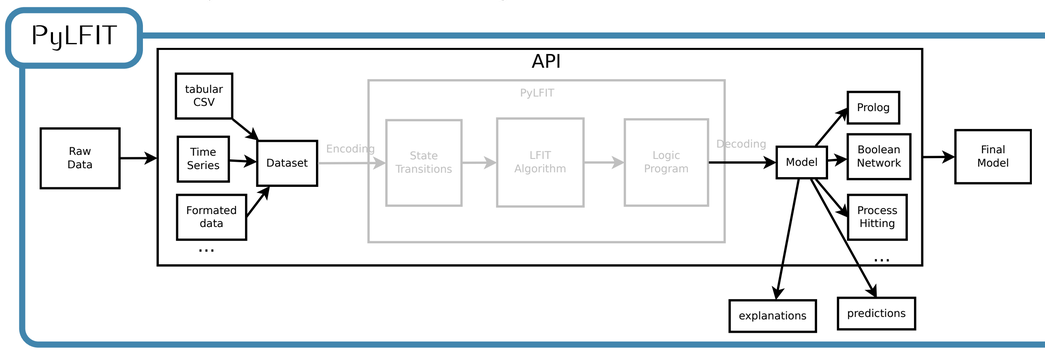
PyLFIT Library

- Open source python library: `pip install pylfit`
- Contain all LFIT algorithms and a simple user API
- Built-in data/model conversion/usage

User API

- Load raw data of different format into a Dataset object
- Choose desired model type and run corresponding LFIT algorithm
- Use model object for predictions, analysis or convert it to other format

PyLFIT



Predictions:

- $\mathcal{DMVLP}$ and $\mathcal{CDMVLP}$ (constraints) can be used for predicting possible target states
- $\mathcal{WDMVLP}$ model both possibility and impossibility, it also adds weights to rules w.r.t. observations to allow probabilistic predictions of target atom occurrence in a transition

$\mathcal{WDMVLP}$

	Likeliness rules	Unlikeliness rules
	$(3, a^0 \leftarrow b^1)$	$(30, a^0 \leftarrow c^1)$
	$(15, a^1 \leftarrow b^0)$	$(5, a^1 \leftarrow c^0)$
	...	...

$predict(a^1, \{a^1, b^1, c^0\} = (0.75, (15, a^1 \leftarrow b^0), (5, a^1 \leftarrow c^0) \longrightarrow$ Likely
$predict(a^0, \{a^1, b^1, c^0\} = (0.09, (3, a^0 \leftarrow b^1), (30, a^0 \leftarrow c^1) \longrightarrow$ Unlikely

- The API provide metrics to evaluate prediction accuracy and quality of explanation rules

Summary

- The polynomiality of **PRIDE** is obtained at the cost of completeness over $P_{\mathcal{O}}(T)$.
- Still, the program learned can reproduce all observations and provides minimal explanation for each of them in the form of optimal rules.
- The source code is available as open source on github and pypi.org (see QR code).
- A user-friendly API allows to easily use LFIT algorithms on different kinds of datasets and is already being used in several research collaborations [4].

[1] Katsumi Inoue, Tony Ribeiro, Chiaki Sakama: Learning from interpretation transition. *Machine Learning* 94(1), 51–79 (2014)
[2] Tony Ribeiro, Maxime Folschette, Morgan Magnin, Olivier Roux, Katsumi Inoue: Learning dynamics with synchronous, asynchronous and general semantics. In: International Conference on Inductive Logic Programming. pp. 118–140. Springer (2018)
[3] Hannes Klarner, Adam Streck, Heike Siebert: PyBoolNet: a Python package for the generation, analysis and visualization of Boolean networks. *Bioinformatics* 33(5), 770–772 (2016).
[4] Alfonso Ortega, Julian Fiérrez, Aythami Morales, Zilong Wang, Tony Ribeiro: Symbolic AI for XAI: Evaluating LFIT Inductive Programming for Fair and Explainable Automatic Recruitment. WACV (Workshops) 2021: 78-87