Learning Dynamics with Synchronous, Asynchronous and General Semantics

Tony Ribeiro[1], Maxime Folschette[2], Morgan Magnin[1], Olivier Roux[1], Katsumi Inoue[3]

1. Laboratoire des Sciences du Numérique de Nantes, France
2. Univ Rennes, Inria, CNRS, IRISA, IRSET, F-35000 Rennes, France
3. National Institute of Informatics, Tokyo, Japan

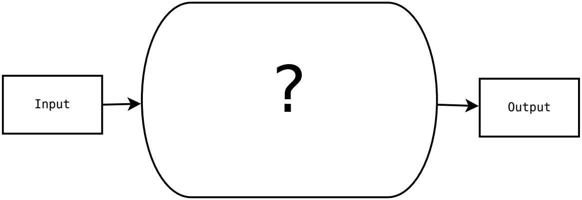
4th September 2018, ILP, Ferarra

Outline

Outline

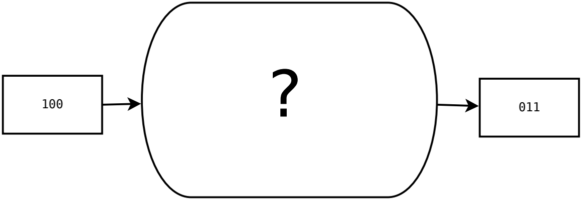
Research area

Idea: given a set of input/output states of a black-box system, learn its internal mechanics.



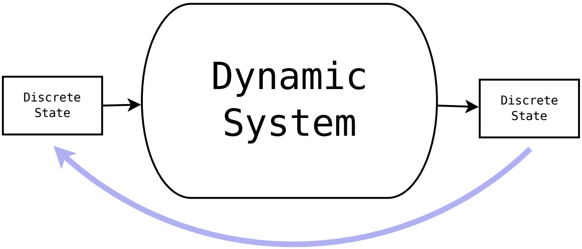
Research area

Discrete system: input/output are vectors of same size which contain discrete values.



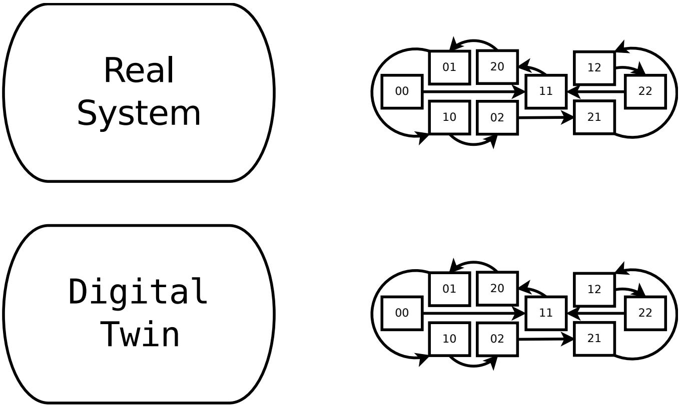
Research area

Dynamic system: input/output are state of the system and output becomes the next input.



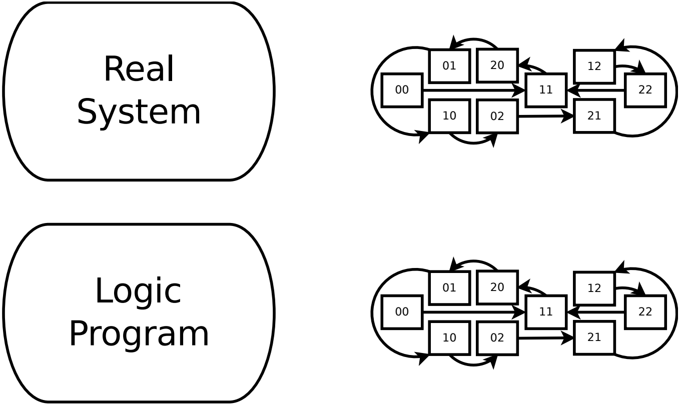
Research area

Goal: produce an artificial system with the same behavior as the one observed, i.e. a digital twin.



Research area

Representation: propositional logic programs with annotated atoms encoding multi-valued variables.

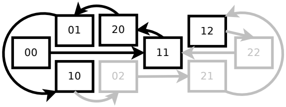


Research area

Method: learn the dynamics of systems from the observations of some of its state transitions.

DATA

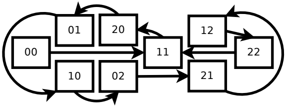
?



RESULTS

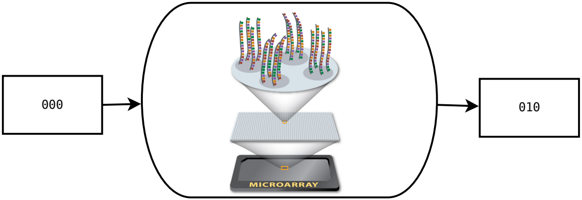
a(0,T) :- a(2,T-1)
a(1,T) :- a(0,T-1), b(0,T-1).
a(2,T) :- a(1,T-1)
a(2,T) :- a(0,T-1), b(2,T-1).

b(0,T) :- a(1,T-1).
b(1,T) :- b(0,T-1).
b(2,T):- b(2,T-1).



Motivation

Data: time series of genes expression levels in a organic cell.
Goal: model genes interactions to understand their influences.

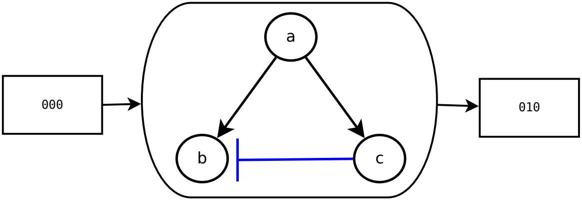


Example (Possible Applications)

- **Bioinformatics**: Construct gene regulatory networks.
- **Robotics**: Learn action models from robot observations.

Motivation

Data: time series of genes expression levels in a organic cell.
Goal: model genes interactions to understand their influences.

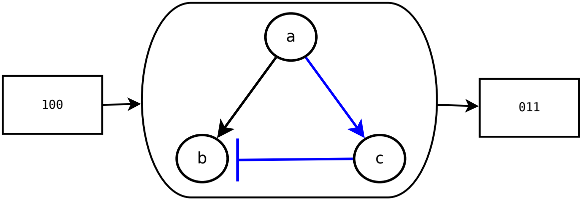


Example (Possible Applications)

- **Bioinformatics**: Construct gene regulatory networks.
- **Robotics**: Learn action models from robot observations.

Motivation

Data: time series of genes expression levels in a organic cell.
Goal: model genes interactions to understand their influences.

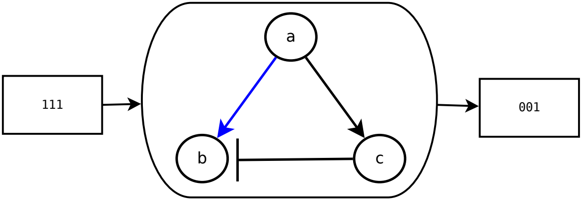


Example (Possible Applications)

- **Bioinformatics**: Construct gene regulatory networks.
- **Robotics**: Learn action models from robot observations.

Motivation

Data: time series of genes expression levels in a organic cell.
Goal: model genes interactions to understand their influences.

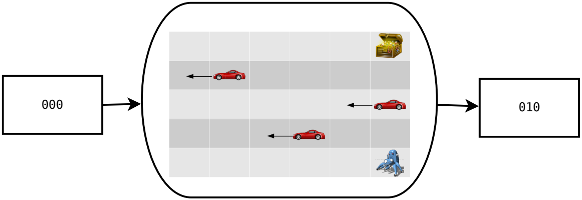


Example (Possible Applications)

- **Bioinformatics**: Construct gene regulatory networks.
- **Robotics**: Learn action models from robot observations.

Motivation

Data: observations of environment evolution according to a robot actions.
Goal: produce a predictive model of the environment for action planning.

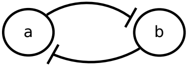


Example (Possible Applications)

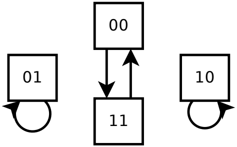
- **Bioinformatics**: Construct gene regulatory networks.
- **Robotics**: Learn action models from robot observations.

Semantics

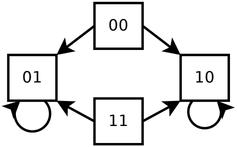
Boolean network transitions differ according to the update semantics used.



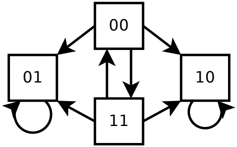
f(a) := not b.
f(b) := not a.



Synchronous



Asynchronous

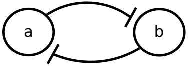


General

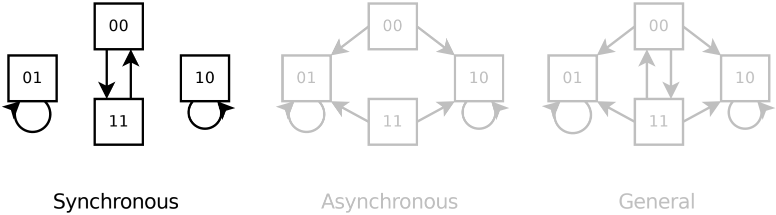
- Synchronous: all variable are updated
- Asynchronous: only one variable is updated
- General: any number of variable can be updated

Semantics

Boolean network transitions differ according to the update semantics used.



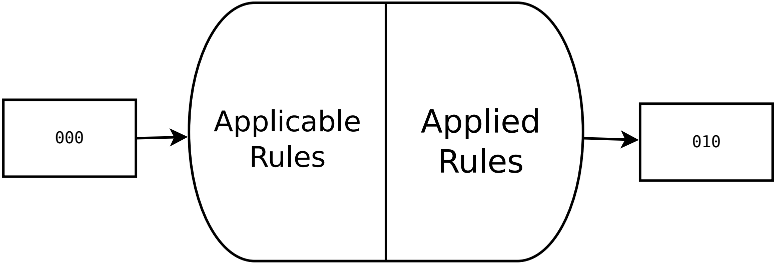
f(a) := not b.
f(b) := not a.



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

What is a semantics?

For those three semantics atleast its about computing the next state by selecting among applicable local rules the ones that will be applied.

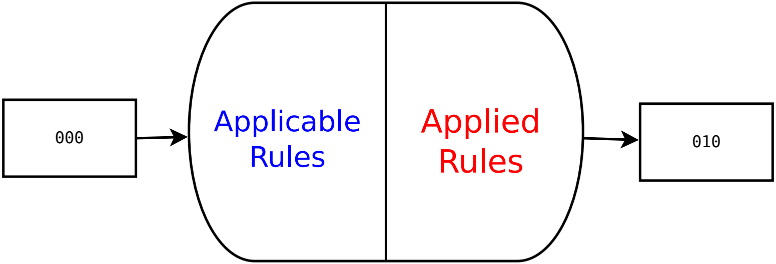


Semantics: what is an applicable rule and what is a valid set of applied rule.

The three semantics differ on the selection but share the same definition of what is an applicable rule.

What is a semantics?

For those three semantics atleast its about computing the next state by selecting among applicable local rules the ones that will be applied.



Semantics: what is an applicable rule and what is a valid set of applied rule.

The three semantics differ on the selection but share the same definition of what is an applicable rule.

Learning algorithm intuition: classification problem

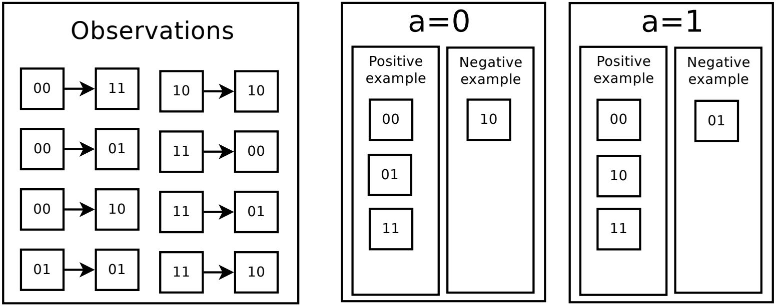
What is an applicable rule?

Learning algorithm intuition: classification problem

What is an applicable rule? The conditions so that a variable can take a certain value in next state.

Learning algorithm intuition: classification problem

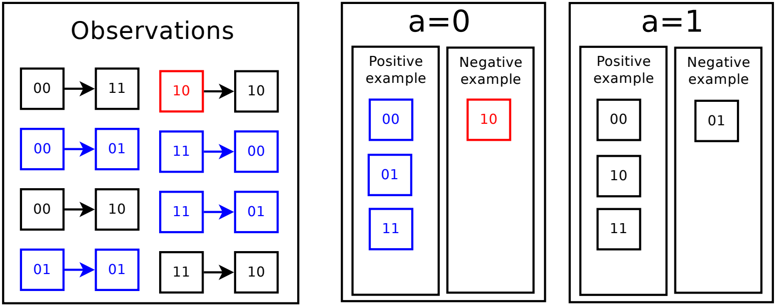
What is an applicable rule? The conditions so that a variable can take a certain value in next state.



Equivalent to a classification problem: for each variable value, what is a typical state where the variable can takes this value in the next state ?

Learning algorithm intuition: classification problem

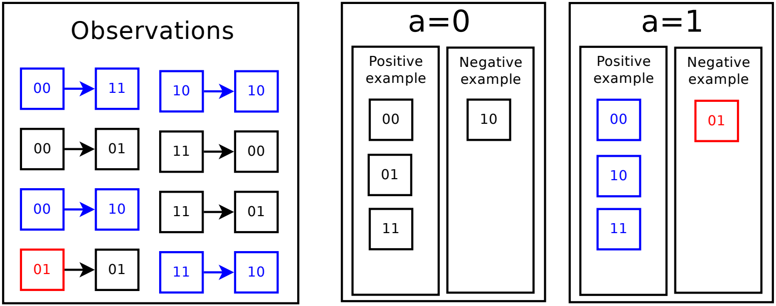
What is an applicable rule? The conditions so that a variable can take a certain value in next state.



Equivalent to a classification problem: for each variable value, what is a typical state where the variable can takes this value in the next state ?

Learning algorithm intuition: classification problem

What is an applicable rule? The conditions so that a variable can take a certain value in next state.



Equivalent to a classification problem: for each variable value, what is a typical state where the variable can takes this value in the next state ?

Outline

Multi-valued Logic ($\mathcal{M}$VL)

Multi-valued Logic ($\mathcal{M}$VL)

Definition (Atoms)

Let $\mathcal{V} = \{\mathrm{v}_1, \dots, \mathrm{v}_n\}$ be a finite set of $n \in \mathbb{N}$ variables, and dom : $\mathcal{V} \to \mathbb{N}$
The atoms of $\mathcal{M}$VL are of the form v^{val} where $\mathrm{v} \in \mathcal{V}$ and
$val \in [\![0; \mathrm{dom}(\mathrm{v})]\!]$. The set of such atoms is denoted by $\mathcal{A}^{\mathcal{V}}_{\mathrm{dom}}$ or simply $\mathcal{A}$.

Multi-valued Logic ($\mathcal{M}$VL)

Definition (Atoms)

Let $\mathcal{V} = \{\mathrm{v}_1, \ldots, \mathrm{v}_n\}$ be a finite set of $n \in \mathbb{N}$ variables, and $\mathrm{dom} : \mathcal{V} \to \mathbb{N}$
The atoms of $\mathcal{M}$VL are of the form v^{val} where $\mathrm{v} \in \mathcal{V}$ and $val \in [\![0; \mathrm{dom}(\mathrm{v})]\!]$. The set of such atoms is denoted by $\mathcal{A}^{\mathcal{V}}_{\mathrm{dom}}$ or simply $\mathcal{A}$.

Definition (Rules)

A $\mathcal{M}$VL rule R is defined by:

$$R = \mathrm{v}_0^{val_0} \leftarrow \mathrm{v}_1^{val_1} \wedge \cdots \wedge \mathrm{v}_m^{val_m} \quad (1)$$

where $\forall i \in [\![0; m]\!], \mathrm{v}_i^{val_i} \in \mathcal{A}$ are atoms in $\mathcal{M}$VL.

Multi-valued Logic ($\mathcal{M}\mathrm{VL}$)

Definition (Atoms)

Let $\mathcal{V} = \{\mathrm{v}_1, \ldots, \mathrm{v}_n\}$ be a finite set of $n \in \mathbb{N}$ variables, and $\mathrm{dom} : \mathcal{V} \to \mathbb{N}$
The atoms of $\mathcal{M}\mathrm{VL}$ are of the form v^{val} where $\mathrm{v} \in \mathcal{V}$ and $val \in [\![0; \mathrm{dom}(\mathrm{v})]\!]$. The set of such atoms is denoted by $\mathcal{A}^{\mathcal{V}}_{\mathrm{dom}}$ or simply $\mathcal{A}$.

Definition (Rules)

A $\mathcal{M}\mathrm{VL}$ rule R is defined by:

$$R = \mathrm{v}_0^{val_0} \leftarrow \mathrm{v}_1^{val_1} \wedge \cdots \wedge \mathrm{v}_m^{val_m} \quad (1)$$

where $\forall i \in [\![0; m]\!], \mathrm{v}_i^{val_i} \in \mathcal{A}$ are atoms in $\mathcal{M}\mathrm{VL}$.

Left-hand side is called the head of R and is denoted $h(R) := \mathrm{v}_0^{val_0}$.

Multi-valued Logic ($\mathcal{M}$VL)

Definition (Atoms)

Let $\mathcal{V} = \{\mathrm{v}_1, \dots, \mathrm{v}_n\}$ be a finite set of $n \in \mathbb{N}$ variables, and dom : $\mathcal{V} \to \mathbb{N}$
The atoms of $\mathcal{M}$VL are of the form v^{val} where $\mathrm{v} \in \mathcal{V}$ and $val \in [\![0; \mathrm{dom}(\mathrm{v})]\!]$. The set of such atoms is denoted by $\mathcal{A}^{\mathcal{V}}_{\mathrm{dom}}$ or simply $\mathcal{A}$.

Definition (Rules)

A $\mathcal{M}$VL rule R is defined by:

$$R \quad = \quad \mathrm{v}_0^{val_0} \leftarrow \mathrm{v}_1^{val_1} \wedge \dots \wedge \mathrm{v}_m^{val_m} \qquad (1)$$

where $\forall i \in [\![0; m]\!], \mathrm{v}_i^{val_i} \in \mathcal{A}$ are atoms in $\mathcal{M}$VL.

$\mathrm{var}(h(R)) := \mathrm{v}_0$ denotes the variable that occurs in $h(R)$.

Multi-valued Logic ($\mathcal{M}$VL)

Definition (Atoms)

Let $\mathcal{V} = \{\mathrm{v}_1, \dots, \mathrm{v}_n\}$ be a finite set of $n \in \mathbb{N}$ variables, and dom : $\mathcal{V} \to \mathbb{N}$
The atoms of $\mathcal{M}$VL are of the form v^{val} where $\mathrm{v} \in \mathcal{V}$ and $val \in [\![0; \mathrm{dom}(\mathrm{v})]\!]$. The set of such atoms is denoted by $\mathcal{A}^{\mathcal{V}}_{\mathrm{dom}}$ or simply $\mathcal{A}$.

Definition (Rules)

A $\mathcal{M}$VL rule R is defined by:

$$R \quad = \quad \mathrm{v}_0^{val_0} \leftarrow \mathrm{v}_1^{val_1} \wedge \dots \wedge \mathrm{v}_m^{val_m} \tag{1}$$

where $\forall i \in [\![0; m]\!], \mathrm{v}_i^{val_i} \in \mathcal{A}$ are atoms in $\mathcal{M}$VL.

Right-hand side is called the body of R, written $b(R) := \{\mathrm{v}_1^{val_1}, \dots, \mathrm{v}_m^{val_m}\}$

Multi-valued Logic ($\mathcal{M}$VL)

Definition (Atoms)

Let $\mathcal{V} = \{v_1, \ldots, v_n\}$ be a finite set of $n \in \mathbb{N}$ variables, and dom : $\mathcal{V} \to \mathbb{N}$
The atoms of $\mathcal{M}$VL are of the form v^{val} where $v \in \mathcal{V}$ and
$val \in [\![0; \mathrm{dom}(v)]\!]$. The set of such atoms is denoted by $\mathcal{A}^{\mathcal{V}}_{\mathrm{dom}}$ or simply $\mathcal{A}$.

Definition (Rules)

A $\mathcal{M}$VL rule R is defined by:

$$R = v_0^{val_0} \leftarrow v_1^{val_1} \wedge \cdots \wedge v_m^{val_m} \tag{1}$$

where $\forall i \in [\![0; m]\!], v_i^{val_i} \in \mathcal{A}$ are atoms in $\mathcal{M}$VL.

A *multi-valued logic program* ($\mathcal{M}$VLP) is a set of $\mathcal{M}$VL rules.

Rules Properties

Rules Properties

Definition (Rule Domination)

Let R_1, R_2 be two $\mathcal{M}\mathrm{VL}$ rules. R_1 dominates R_2, written $R_2 \leq R_1$ if $h(R_1) = h(R_2)$ and $b(R_1) \subseteq b(R_2)$.

Rules Properties

Definition (Rule Domination)

Let R_1, R_2 be two $\mathcal{M}\mathrm{VL}$ rules. R_1 dominates R_2, written $R_2 \leq R_1$ if $h(R_1) = h(R_2)$ and $b(R_1) \subseteq b(R_2)$.

Proposition (Double domination is equality)

If $R_1 \leq R_2$ *and* $R_2 \leq R_1$ *then* $R_1 = R_2$.

Rules with the most general bodies dominate the other rules.
These are the rules we want since they cover the most general cases.

Dynamical System Modeling

Dynamical System Modeling

Definition (Discrete State)

A discrete state s is a function from $\mathcal{V}$ to $\mathbb{N}$, i.e., it associates an integer value to each variable in $\mathcal{V}$.

Dynamical System Modeling

Definition (Discrete State)

A discrete state s is a function from $\mathcal{V}$ to $\mathbb{N}$, i.e., it associates an integer value to each variable in $\mathcal{V}$.

It can be equivalently represented by the set of atoms $\{\mathrm{v}^{s(\mathrm{v})} \mid \mathrm{v} \in \mathcal{V}\}$ and thus we can use classical set operations on it.

Dynamical System Modeling

Definition (Discrete State)

A discrete state s is a function from $\mathcal{V}$ to $\mathbb{N}$, i.e., it associates an integer value to each variable in $\mathcal{V}$.

It can be equivalently represented by the set of atoms $\{\mathrm{v}^{s(\mathrm{v})} \mid \mathrm{v} \in \mathcal{V}\}$ and thus we can use classical set operations on it.

Definition (Transitions)

We write $\mathcal{S}$ to denote the set of all discrete states, and a couple of states $(s, s') \in \mathcal{S}^2$ is called a transition.

Dynamical System Modeling

Dynamical System Modeling

Definition (Rule-state matching)

Let $s \in \mathcal{S}$. The $\mathcal{M}\mathrm{VL}$ rule R matches s, written $R \sqcap s$, if $b(R) \subseteq s$.

When matching a state, a rule can be used to realize a transition.

Dynamical System Modeling

Definition (Rule-state matching)

Let $s \in \mathcal{S}$. The $\mathcal{M}\mathrm{VL}$ rule R matches s, written $R \sqcap s$, if $b(R) \subseteq s$.

When matching a state, a rule can be used to realize a transition.

Definition (Rule realization)

A rule R realizes the transition (s, s'), written $s \xrightarrow{R} s'$, if $R \sqcap s, h(R) \in s'$.

Dynamical System Modeling

Definition (Rule-state matching)

Let $s \in \mathcal{S}$. The $\mathcal{M}\mathrm{VL}$ rule R matches s, written $R \sqcap s$, if $b(R) \subseteq s$.

When matching a state, a rule can be used to realize a transition.

Definition (Rule realization)

A rule R realizes the transition (s, s'), written $s \xrightarrow{R} s'$, if $R \sqcap s, h(R) \in s'$.

Definition (Program realization)

A $\mathcal{M}\mathrm{VLP}$ P realizes (s, s'), written $s \xrightarrow{P} s'$, if
$\forall \mathrm{v} \in \mathcal{V}, \exists R \in P, \mathrm{var}(h(R)) = \mathrm{v} \wedge s \xrightarrow{R} s'$.
It realizes $T \subseteq \mathcal{S}^2$, written $\overset{P}{\hookrightarrow} T$, if $\forall (s, s') \in T, s \xrightarrow{P} s'$.

Desired Properties

In the following, for all sets of transitions $T \subseteq \mathcal{S}^2$, we denote:
$\mathrm{fst}(T) := \{s \in \mathcal{S} \mid \exists (s_1, s_2) \in T, s_1 = s\}$.

Desired Properties

In the following, for all sets of transitions $T \subseteq \mathcal{S}^2$, we denote:
$\mathrm{fst}(T) := \{s \in \mathcal{S} \mid \exists (s_1, s_2) \in T, s_1 = s\}$.

Definition (Conflicts)

A $\mathcal{M}\mathrm{VL}$ rule R conflicts with a set of transitions $T \subseteq \mathcal{S}^2$ when $\exists s \in \mathrm{fst}(T), \big(R \sqcap s \land \forall (s, s') \in T, h(R) \notin s'\big)$.

Desired Properties

In the following, for all sets of transitions $T \subseteq \mathcal{S}^2$, we denote:
$\mathrm{fst}(T) := \{s \in \mathcal{S} \mid \exists (s_1, s_2) \in T, s_1 = s\}$.

Definition (Conflicts)

A $\mathcal{M}\mathrm{VL}$ rule R conflicts with a set of transitions $T \subseteq \mathcal{S}^2$ when
$\exists s \in \mathrm{fst}(T), (R \sqcap s \wedge \forall (s, s') \in T, h(R) \notin s')$.

Definition (Concurrent rules)

Two $\mathcal{M}\mathrm{VL}$ rules R and R' are concurrent when
$R \sqcap R' \wedge \mathrm{var}(h(R)) = \mathrm{var}(h(R')) \wedge h(R) \neq h(R')$.

Definition (Complete program)

A $\mathcal{M}\mathrm{VLP}$ P is complete if $\forall s \in \mathcal{S}, \forall \mathrm{v} \in \mathcal{V}, \exists R \in P, R \sqcap s \wedge \mathrm{var}(h(R)) = \mathrm{v}$.

Definition (Complete program)

A $\mathcal{M}$VLP P is complete if $\forall s \in \mathcal{S}, \forall \mathrm{v} \in \mathcal{V}, \exists R \in P, R \sqcap s \wedge \mathrm{var}(h(R)) = \mathrm{v}$.

A complete program realize atleast one transition for each state $s \in \mathcal{S}$.

Definition (Complete program)

A $\mathcal{M}\mathrm{VLP}$ P is complete if $\forall s \in \mathcal{S}, \forall \mathrm{v} \in \mathcal{V}, \exists R \in P, R \sqcap s \wedge \mathrm{var}(h(R)) = \mathrm{v}$.

A complete program realize atleast one transition for each state $s \in \mathcal{S}$.

Definition (Consistent program)

A $\mathcal{M}\mathrm{VLP}$ P is consistent with a set of transitions T if P does not contains any rule R conflicting with T.

Definition (Complete program)

A $\mathcal{M}$VLP P is complete if $\forall s \in \mathcal{S}, \forall \mathrm{v} \in \mathcal{V}, \exists R \in P, R \sqcap s \wedge \mathrm{var}(h(R)) = \mathrm{v}$.

A complete program realize atleast one transition for each state $s \in \mathcal{S}$.

Definition (Consistent program)

A $\mathcal{M}$VLP P is consistent with a set of transitions T if P does not contains any rule R conflicting with T.

Let $s \in \mathrm{fst}(T)$, a program consistent with T will only realize the transitions $(s, s') \in T$.

Optimal $\mathcal{M}$VLP

Optimal $\mathcal{M}$VLP

Definition (Suitable program)

Let $T \subseteq \mathcal{S}^2$. A $\mathcal{M}$VLP P is suitable for T when:

- P is consistent with T, — Cover no negative example
- P realizes T, — Cover all positive example
- P is complete — Cover all state space
- $\forall R$ not conflicting with T, $\exists R' \in P$ s.t. $R \leq R'$. — Cover all hypotheses

Optimal $\mathcal{M}$VLP

Definition (Suitable program)

Let $T \subseteq \mathcal{S}^2$. A $\mathcal{M}$VLP P is suitable for T when:

- P is consistent with T, Cover no negative example
- P realizes T, Cover all positive example
- P is complete Cover all state space
- $\forall R$ not conflicting with T, $\exists R' \in P$ s.t. $R \leq R'$. Cover all hypotheses

Definition (Optimal program)

If in addition, $\forall R \in P$, all the rules R' belonging to a $\mathcal{M}$VLP suitable for T are such that $R \leq R'$ implies $R' \leq R$ then P is called optimal.

An optimal program is the set of all rules that are not dominated by any consistent rules. Contains all minimal hypotheses

Optimal $\mathcal{M}$VLP

Optimal $\mathcal{M}$VLP

Proposition

Let $T \subseteq \mathcal{S}^2$. The $\mathcal{M}$VLP optimal for T is unique and denoted $P_{\mathcal{O}}(T)$.

Optimal $\mathcal{M}$VLP

Proposition

Let $T \subseteq \mathcal{S}^2$. The $\mathcal{M}$VLP optimal for T is unique and denoted $P_{\mathcal{O}}(T)$.

Troll mode on: does it works if it is the empty set? :p

Optimal $\mathcal{M}$VLP

Proposition

Let $T \subseteq \mathcal{S}^2$*. The* $\mathcal{M}$VLP *optimal for* T *is unique and denoted* $P_{\mathcal{O}}(T)$*.*

Troll mode on: does it works if it is the empty set? :p

Proposition

$P_{\mathcal{O}}(\emptyset) = \{\mathrm{v}^{val} \leftarrow \emptyset \mid \mathrm{v}^{val} \in \mathcal{A}\}$.

Optimal $\mathcal{M}\mathrm{VLP}$

Proposition

Let $T \subseteq \mathcal{S}^2$. The $\mathcal{M}\mathrm{VLP}$ optimal for T is unique and denoted $P_{\mathcal{O}}(T)$.

Troll mode on: does it works if it is the empty set? :p

Proposition

$P_{\mathcal{O}}(\emptyset) = \{\mathrm{v}^{val} \leftarrow \emptyset \mid \mathrm{v}^{val} \in \mathcal{A}\}$.

Yeah ! And this property is the starting point of the learning algorithm.

Optimal $\mathcal{M}$VLP

Proposition

Let $T \subseteq \mathcal{S}^2$. The $\mathcal{M}$VLP optimal for T is unique and denoted $P_{\mathcal{O}}(T)$.

Troll mode on: does it works if it is the empty set? :p

Proposition

$P_{\mathcal{O}}(\emptyset) = \{\mathrm{v}^{val} \leftarrow \emptyset \mid \mathrm{v}^{val} \in \mathcal{A}\}$.

Yeah ! And this property is the starting point of the learning algorithm.

Proposition

Let $T \subseteq \mathcal{S}^2$. If P is a $\mathcal{M}$VLP suitable for T, then
$P_{\mathcal{O}}(T) = \{R \in P \mid \forall R' \in P, R \leq R' \implies R' \leq R\}$

Optimal $\mathcal{M}$VLP

Proposition

Let $T \subseteq \mathcal{S}^2$. The $\mathcal{M}$VLP optimal for T is unique and denoted $P_{\mathcal{O}}(T)$.

Troll mode on: does it works if it is the empty set? :p

Proposition

$P_{\mathcal{O}}(\emptyset) = \{\mathrm{v}^{val} \leftarrow \emptyset \mid \mathrm{v}^{val} \in \mathcal{A}\}$.

Yeah ! And this property is the starting point of the learning algorithm.

Proposition

Let $T \subseteq \mathcal{S}^2$. If P is a $\mathcal{M}$VLP suitable for T, then
$P_{\mathcal{O}}(T) = \{R \in P \mid \forall R' \in P, R \leq R' \implies R' \leq R\}$

We can obtain the optimal program from any suitable program by simply removing the dominated rules.

Outline

Learning Process

Learning Process

How to make a minimal modifications of a $\mathcal{M}\mathrm{VLP}$ in order to be suitable with a new set of transitions?

Learning Process

How to make a minimal modifications of a $\mathcal{M}\mathrm{VLP}$ in order to be suitable with a new set of transitions?

Definition (Rule least specialization)

Let R be a $\mathcal{M}\mathrm{VL}$ rule and $s \in \mathcal{S}$ such that $R \sqcap s$. The least specialization of R by s is:

$$L_{\mathrm{spe}}(R, s) := \{h(R) \leftarrow b(R) \cup \{\mathrm{v}^{val}\} \mid \mathrm{v}^{val} \in \mathcal{A} \wedge \mathrm{v}^{val} \notin s \wedge \forall val' \in \mathbb{N}, \mathrm{v}^{val'} \notin b(R)\}.$$

Learning Process

How to make a minimal modifications of a $\mathcal{M}\mathrm{VLP}$ in order to be suitable with a new set of transitions?

Definition (Rule least specialization)

Let R be a $\mathcal{M}\mathrm{VL}$ rule and $s \in \mathcal{S}$ such that $R \sqcap s$. The least specialization of R by s is:

$$L_{\mathrm{spe}}(R, s) := \{h(R) \leftarrow b(R) \cup \{\mathrm{v}^{val}\} \mid \mathrm{v}^{val} \in \mathcal{A} \land \mathrm{v}^{val} \notin s \land \forall val' \in \mathbb{N}, \mathrm{v}^{val'} \notin b(R)\}.$$

Thus, a $\mathcal{M}\mathrm{VLP}$ can be revised to only realizes given transitions from s.

Learning Process

How to make a minimal modifications of a $\mathcal{M}\mathrm{VLP}$ in order to be suitable with a new set of transitions?

Definition (Rule least specialization)

Let R be a $\mathcal{M}\mathrm{VL}$ rule and $s \in \mathcal{S}$ such that $R \sqcap s$. The least specialization of R by s is:

$$L_{\mathrm{spe}}(R, s) := \{h(R) \leftarrow b(R) \cup \{\mathrm{v}^{val}\} \mid \mathrm{v}^{val} \in \mathcal{A} \land \mathrm{v}^{val} \notin s \land \forall val' \in \mathbb{N}, \mathrm{v}^{val'} \notin b(R)\}.$$

Thus, a $\mathcal{M}\mathrm{VLP}$ can be revised to only realizes given transitions from s.

Definition (Program least revision)

Let P be a $\mathcal{M}\mathrm{VLP}$, $s \in \mathcal{S}$ and $T \subseteq \mathcal{S}^2$ such that $\mathrm{fst}(T) = \{s\}$. Let $R_P := \{R \in P \mid R \text{ conflicts with } T\}$. The least revision of P by T is $L_{\mathrm{rev}}(P, T) := (P \setminus R_P) \cup \bigcup_{R \in R_P} L_{\mathrm{spe}}(R, s)$.

Learning Process

Learning Process

Guess what? Least revision can conserves suitability :)

Learning Process

Guess what? Least revision can conserves suitability :)

Theorem

Let $s \in \mathcal{S}$ *and* $T, T' \subseteq \mathcal{S}^2$ *such that* $|\mathrm{fst}(T')| = 1 \land \mathrm{fst}(T) \cap \mathrm{fst}(T') = \emptyset$. $L_{\mathrm{rev}}(P_{\mathcal{O}}(T), T')$ *is a* $\mathcal{M}\mathrm{VLP}$ *suitable for* $T \cup T'$.

Learning Process

Guess what? Least revision can conserves suitability :)

Theorem

Let $s \in \mathcal{S}$ and $T, T' \subseteq \mathcal{S}^2$ such that $|\mathrm{fst}(T')| = 1 \wedge \mathrm{fst}(T) \cap \mathrm{fst}(T') = \emptyset$. $L_{\mathrm{rev}}(P_{\mathcal{O}}(T), T')$ is a $\mathcal{M}\mathrm{VLP}$ suitable for $T \cup T'$.

In association with previous results it gives a method to iteratively compute $P_{\mathcal{O}}(T)$ for any $T \subseteq \mathcal{S}^2$, starting from $P_{\mathcal{O}}(\emptyset)$.

GULA: General Usage LFIT Algorithm

GULA: General Usage LFIT Algorithm

GULA:

INPUT: a set of atoms $\mathcal{A}$ and a set of transitions $T \subseteq \mathcal{S}^2$.

GULA: General Usage LFIT Algorithm

GULA:

INPUT: a set of atoms $\mathcal{A}$ and a set of transitions $T \subseteq \mathcal{S}^2$.

For each atom $\mathrm{v}^{val} \in \mathcal{A}$

GULA: General Usage LFIT Algorithm

GULA:

INPUT: a set of atoms $\mathcal{A}$ and a set of transitions $T \subseteq \mathcal{S}^2$.

For each atom $\mathrm{v}^{val} \in \mathcal{A}$

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$

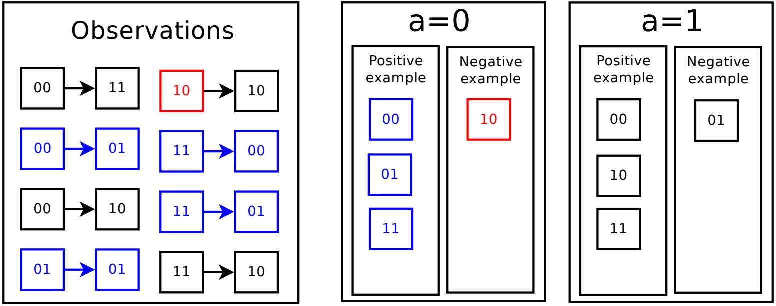
GULA: General Usage LFIT Algorithm

GULA:

INPUT: a set of atoms $\mathcal{A}$ and a set of transitions $T \subseteq \mathcal{S}^2$.

For each atom $\mathrm{v}^{val} \in \mathcal{A}$

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$



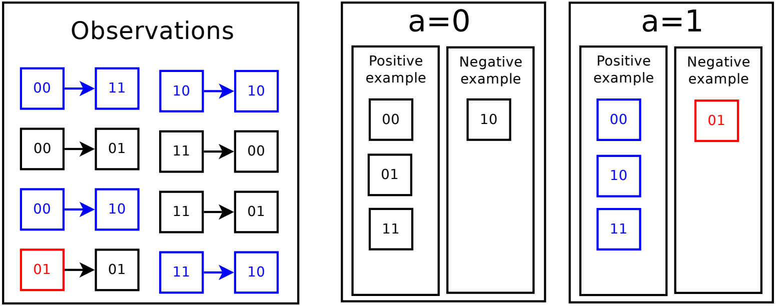
GULA: General Usage LFIT Algorithm

GULA:

INPUT: a set of atoms $\mathcal{A}$ and a set of transitions $T \subseteq \mathcal{S}^2$.

For each atom $\mathrm{v}^{val} \in \mathcal{A}$

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$



GULA: General Usage LFIT Algorithm

GULA:

INPUT: a set of atoms $\mathcal{A}$ and a set of transitions $T \subseteq \mathcal{S}^2$.

For each atom $\mathrm{v}^{val} \in \mathcal{A}$

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \emptyset\}$

GULA: General Usage LFIT Algorithm

GULA:

INPUT: a set of atoms $\mathcal{A}$ and a set of transitions $T \subseteq \mathcal{S}^2$.

For each atom $\mathrm{v}^{val} \in \mathcal{A}$

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \emptyset\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$

GULA: General Usage LFIT Algorithm

GULA:

INPUT: a set of atoms $\mathcal{A}$ and a set of transitions $T \subseteq \mathcal{S}^2$.

For each atom $\mathrm{v}^{val} \in \mathcal{A}$

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \emptyset\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Extract each rule R of $P_{\mathrm{v}^{val}}$ that matches s:
    $M_{\mathrm{v}^{val}} := \{R \in P \mid b(R) \subseteq s\}, P_{\mathrm{v}^{val}} := P_{\mathrm{v}^{val}} \setminus M_{\mathrm{v}^{val}}$.

GULA: General Usage LFIT Algorithm

GULA:

INPUT: a set of atoms $\mathcal{A}$ and a set of transitions $T \subseteq \mathcal{S}^2$.

For each atom $\mathrm{v}^{val} \in \mathcal{A}$

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \emptyset\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Extract each rule R of $P_{\mathrm{v}^{val}}$ that matches s:
    $M_{\mathrm{v}^{val}} := \{R \in P \mid b(R) \subseteq s\}, P_{\mathrm{v}^{val}} := P_{\mathrm{v}^{val}} \setminus M_{\mathrm{v}^{val}}$.
  - For each rule $R \in M_{\mathrm{v}^{val}}$

GULA: General Usage LFIT Algorithm

GULA:

INPUT: a set of atoms $\mathcal{A}$ and a set of transitions $T \subseteq \mathcal{S}^2$.

For each atom $\mathrm{v}^{val} \in \mathcal{A}$

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \emptyset\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Extract each rule R of $P_{\mathrm{v}^{val}}$ that matches s:
    $M_{\mathrm{v}^{val}} := \{R \in P \mid b(R) \subseteq s\}, P_{\mathrm{v}^{val}} := P_{\mathrm{v}^{val}} \setminus M_{\mathrm{v}^{val}}$.
  - For each rule $R \in M_{\mathrm{v}^{val}}$
    - ⋆ Compute its least specialization $P' = L_{\mathrm{spe}}(R, s)$.

GULA: General Usage LFIT Algorithm

GULA:

INPUT: a set of atoms $\mathcal{A}$ and a set of transitions $T \subseteq \mathcal{S}^2$.

For each atom $\mathrm{v}^{val} \in \mathcal{A}$

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \emptyset\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Extract each rule R of $P_{\mathrm{v}^{val}}$ that matches s:
    $M_{\mathrm{v}^{val}} := \{R \in P \mid b(R) \subseteq s\}, P_{\mathrm{v}^{val}} := P_{\mathrm{v}^{val}} \setminus M_{\mathrm{v}^{val}}$.
  - For each rule $R \in M_{\mathrm{v}^{val}}$
    - Compute its least specialization $P' = L_{\mathrm{spe}}(R, s)$.
    - Remove all the rules in P' dominated by a rule in $P_{\mathrm{v}^{val}}$.

GULA: General Usage LFIT Algorithm

GULA:

INPUT: a set of atoms $\mathcal{A}$ and a set of transitions $T \subseteq \mathcal{S}^2$.

For each atom $\mathrm{v}^{val} \in \mathcal{A}$

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \emptyset\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Extract each rule R of $P_{\mathrm{v}^{val}}$ that matches s:
    $M_{\mathrm{v}^{val}} := \{R \in P \mid b(R) \subseteq s\}, P_{\mathrm{v}^{val}} := P_{\mathrm{v}^{val}} \setminus M_{\mathrm{v}^{val}}$.
  - For each rule $R \in M_{\mathrm{v}^{val}}$
    - Compute its least specialization $P' = L_{\mathrm{spe}}(R, s)$.
    - Remove all the rules in P' dominated by a rule in $P_{\mathrm{v}^{val}}$.
    - Remove all the rules in $P_{\mathrm{v}^{val}}$ dominated by a rule in P'.

GULA: General Usage LFIT Algorithm

GULA:

INPUT: a set of atoms $\mathcal{A}$ and a set of transitions $T \subseteq \mathcal{S}^2$.

For each atom $\mathrm{v}^{val} \in \mathcal{A}$

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \emptyset\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Extract each rule R of $P_{\mathrm{v}^{val}}$ that matches s:
    $M_{\mathrm{v}^{val}} := \{R \in P \mid b(R) \subseteq s\}, P_{\mathrm{v}^{val}} := P_{\mathrm{v}^{val}} \setminus M_{\mathrm{v}^{val}}$.
  - For each rule $R \in M_{\mathrm{v}^{val}}$
    - Compute its least specialization $P' = L_{\mathrm{spe}}(R, s)$.
    - Remove all the rules in P' dominated by a rule in $P_{\mathrm{v}^{val}}$.
    - Remove all the rules in $P_{\mathrm{v}^{val}}$ dominated by a rule in P'.
    - Add all remaining rules in P' to $P_{\mathrm{v}^{val}}$.

GULA: General Usage LFIT Algorithm

GULA:

INPUT: a set of atoms $\mathcal{A}$ and a set of transitions $T \subseteq \mathcal{S}^2$.

For each atom $\mathrm{v}^{val} \in \mathcal{A}$

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists(s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \emptyset\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Extract each rule R of $P_{\mathrm{v}^{val}}$ that matches s:
    $M_{\mathrm{v}^{val}} := \{R \in P \mid b(R) \subseteq s\}, P_{\mathrm{v}^{val}} := P_{\mathrm{v}^{val}} \setminus M_{\mathrm{v}^{val}}$.
  - For each rule $R \in M_{\mathrm{v}^{val}}$
    - Compute its least specialization $P' = L_{\mathrm{spe}}(R, s)$.
    - Remove all the rules in P' dominated by a rule in $P_{\mathrm{v}^{val}}$.
    - Remove all the rules in $P_{\mathrm{v}^{val}}$ dominated by a rule in P'.
    - Add all remaining rules in P' to $P_{\mathrm{v}^{val}}$.
- $P := P \cup P_{\mathrm{v}^{val}}$

GULA: General Usage LFIT Algorithm

GULA:

INPUT: a set of atoms $\mathcal{A}$ and a set of transitions $T \subseteq \mathcal{S}^2$.

For each atom $\mathrm{v}^{val} \in \mathcal{A}$

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \emptyset\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Extract each rule R of $P_{\mathrm{v}^{val}}$ that matches s:
    $M_{\mathrm{v}^{val}} := \{R \in P \mid b(R) \subseteq s\}, P_{\mathrm{v}^{val}} := P_{\mathrm{v}^{val}} \setminus M_{\mathrm{v}^{val}}$.
  - For each rule $R \in M_{\mathrm{v}^{val}}$
    - Compute its least specialization $P' = L_{\mathrm{spe}}(R, s)$.
    - Remove all the rules in P' dominated by a rule in $P_{\mathrm{v}^{val}}$.
    - Remove all the rules in $P_{\mathrm{v}^{val}}$ dominated by a rule in P'.
    - Add all remaining rules in P' to $P_{\mathrm{v}^{val}}$.
- $P := P \cup P_{\mathrm{v}^{val}}$

OUTPUT: $P_{\mathcal{O}}(T) := P$.

Outline

Where is the semantics gone?

Where is the semantics gone?

The formalization of $\mathcal{M}\mathrm{VLP}$ is independant of the semantics that produced its transitions.

Where is the semantics gone?

The formalization of $\mathcal{M}\mathrm{VLP}$ is independant of the semantics that produced its transitions.

Definition (Semantics)

Let $\mathcal{A}^{\mathcal{V}}_{\mathsf{dom}}$ be a set of atoms and $\mathcal{S}$ the corresponding set of states. A semantics (on $\mathcal{A}^{\mathcal{V}}_{\mathsf{dom}}$) is a function that associates, to each complete $\mathcal{M}\mathrm{VLP}$ P, a set of transitions $T \subseteq \mathcal{S}^2$ so that: $\mathrm{fst}(T) = \mathcal{S}$. Equivalently, a semantics can be seen as a function of $(\mathrm{c}\text{-}\mathcal{M}\mathrm{VLP} \rightarrow (\mathcal{S} \rightarrow \wp(\mathcal{S}) \setminus \emptyset))$ where $\mathrm{c}\text{-}\mathcal{M}\mathrm{VLP}$ is the set of complete $\mathcal{M}\mathrm{VLP}$s and $\wp$ is the power set operator.

Definition (Synchronous semantics)

The synchronous semantics $\mathcal{T}_{syn}$ is defined by:

$$\mathcal{T}_{syn} : P \mapsto \{(s, s') \in \mathcal{S}^2 \mid s' \subseteq \{h(R) \in \mathcal{A} \mid R \in P, R \sqcap s\}\}$$

Definition (Synchronous semantics)

The synchronous semantics $\mathcal{T}_{syn}$ is defined by:

$$\mathcal{T}_{syn} : P \mapsto \{(s, s') \in \mathcal{S}^2 \mid s' \subseteq \{h(R) \in \mathcal{A} \mid R \in P, R \sqcap s\}\}$$

Definition (Asynchronous semantics)

The asynchronous semantics $\mathcal{T}_{asyn}$ is defined by:

$$\mathcal{T}_{asyn} : P \mapsto \{(s, s \setminus\!\!\setminus \{h(R)\}) \in \mathcal{S}^2 \mid R \in P \wedge R \sqcap s \wedge h(R) \notin s\} \\ \cup \{(s, s) \in \mathcal{S}^2 \mid \forall R \in P, R \sqcap s \implies h(R) \in s\}.$$

Definition (Synchronous semantics)

The synchronous semantics $\mathcal{T}_{syn}$ is defined by:

$$\mathcal{T}_{syn} : P \mapsto \{(s, s') \in \mathcal{S}^2 \mid s' \subseteq \{h(R) \in \mathcal{A} \mid R \in P, R \sqcap s\}\}$$

Definition (Asynchronous semantics)

The asynchronous semantics $\mathcal{T}_{asyn}$ is defined by:

$$\mathcal{T}_{asyn} : P \mapsto \{(s, s \setminus\setminus \{h(R)\}) \in \mathcal{S}^2 \mid R \in P \wedge R \sqcap s \wedge h(R) \notin s\}$$
$$\cup \{(s, s) \in \mathcal{S}^2 \mid \forall R \in P, R \sqcap s \implies h(R) \in s\}.$$

Definition (General semantics)

The general semantics $\mathcal{T}_{gen}$ is defined by:

$$\mathcal{T}_{gen} : P \mapsto \{(s, s \setminus\setminus r) \in \mathcal{S}^2 \mid r \subseteq \{h(R) \in \mathcal{A} \mid R \in P \wedge R \sqcap s\} \wedge$$
$$\forall \mathrm{v}_1^{val_1}, \mathrm{v}_2^{val_2} \in r, \mathrm{v}_1 = \mathrm{v}_2 \implies val_1 = val_2\}.$$

Definition (Synchronous semantics)

The synchronous semantics $\mathcal{T}_{syn}$ is defined by:

$$\mathcal{T}_{syn} : P \mapsto \{(s, s') \in \mathcal{S}^2 \mid s' \subseteq \{h(R) \in \mathcal{A} \mid R \in P, R \sqcap s\}\}$$

Definition (Asynchronous semantics)

The asynchronous semantics $\mathcal{T}_{asyn}$ is defined by:

$$\mathcal{T}_{asyn} : P \mapsto \{(s, s \setminus\setminus \{h(R)\}) \in \mathcal{S}^2 \mid R \in P \wedge R \sqcap s \wedge h(R) \notin s\}$$
$$\cup \{(s, s) \in \mathcal{S}^2 \mid \forall R \in P, R \sqcap s \implies h(R) \in s\}.$$

Definition (General semantics)

The general semantics $\mathcal{T}_{gen}$ is defined by:

$$\mathcal{T}_{gen} : P \mapsto \{(s, s \setminus\setminus r) \in \mathcal{S}^2 \mid r \subseteq \{h(R) \in \mathcal{A} \mid R \in P \wedge R \sqcap s\} \wedge$$
$$\forall \mathrm{v}_1^{val_1}, \mathrm{v}_2^{val_2} \in r, \mathrm{v}_1 = \mathrm{v}_2 \implies val_1 = val_2\}.$$

Semantic free modeling

Finally, we can state that the definitions and method developed in the previous section are independent of those three semantics.

Theorem (Semantics-free correctness)

Let P be a $\mathcal{M}\mathrm{VLP}$ such that P is complete.

- $\mathcal{T}_{syn}(P) = \mathcal{T}_{syn}(P_{\mathcal{O}}(\mathcal{T}_{syn}(P)))$,
- $\mathcal{T}_{asyn}(P) = \mathcal{T}_{asyn}(P_{\mathcal{O}}(\mathcal{T}_{asyn}(P)))$,
- $\mathcal{T}_{gen}(P) = \mathcal{T}_{gen}(P_{\mathcal{O}}(\mathcal{T}_{gen}(P)))$.

Semantic free modeling

Finally, we can state that the definitions and method developed in the previous section are independent of those three semantics.

Theorem (Semantics-free correctness)

Let P be a $\mathcal{M}$VLP such that P is complete.

- $\mathcal{T}_{syn}(P) = \mathcal{T}_{syn}(P_{\mathcal{O}}(\mathcal{T}_{syn}(P)))$,
- $\mathcal{T}_{asyn}(P) = \mathcal{T}_{asyn}(P_{\mathcal{O}}(\mathcal{T}_{asyn}(P)))$,
- $\mathcal{T}_{gen}(P) = \mathcal{T}_{gen}(P_{\mathcal{O}}(\mathcal{T}_{gen}(P)))$.

Whatever the semantic which produced T, given the optimal $\mathcal{M}$VLP of T we can reproduce exactly T with the same semantic.

GULA: General Usage LFIT Algorithm

And **GULA** can learn such an optimal $\mathcal{M}\mathrm{VLP}$ from T.

Theorem (GULA Termination, soundness, completeness, optimality)

Let $T \subseteq \mathcal{S}^2$. The call **GULA**$(\mathcal{A}, T)$ *terminates and* **GULA**$(\mathcal{A}, T) = P_{\mathcal{O}}(T)$.

GULA: General Usage LFIT Algorithm

And **GULA** can learn such an optimal $\mathcal{M}\mathrm{VLP}$ from T.

Theorem (GULA Termination, soundness, completeness, optimality)

Let $T \subseteq \mathcal{S}^2$. The call **GULA**$(\mathcal{A}, T)$ *terminates and* **GULA**$(\mathcal{A}, T) = P_{\mathcal{O}}(T)$.

Making the algorithm semantic-free atleast for those three semantics.

GULA: General Usage LFIT Algorithm

And **GULA** can learn such an optimal $\mathcal{M}\mathrm{VLP}$ from T.

Theorem (**GULA** Termination, soundness, completeness, optimality)

Let $T \subseteq \mathcal{S}^2$*. The call* **GULA**$(\mathcal{A}, T)$ *terminates and* **GULA**$(\mathcal{A}, T) = P_{\mathcal{O}}(T)$.

Making the algorithm semantic-free atleast for those three semantics.

Theorem (Semantic-freeness)

Let P *be a* $\mathcal{M}\mathrm{VLP}$ *such that* P *is complete.*

- **GULA**$(\mathcal{A}, \mathcal{T}_{syn}(P)) = P_{\mathcal{O}}(\mathcal{T}_{syn}(P))$
- **GULA**$(\mathcal{A}, \mathcal{T}_{asyn}(P)) = P_{\mathcal{O}}(\mathcal{T}_{asyn}(P))$
- **GULA**$(\mathcal{A}, \mathcal{T}_{gen}(P)) = P_{\mathcal{O}}(\mathcal{T}_{gen}(P))$

GULA: General Usage LFIT Algorithm

And **GULA** can learn such an optimal $\mathcal{M}\mathrm{VLP}$ from T.

Theorem (GULA Termination, soundness, completeness, optimality)

Let $T \subseteq \mathcal{S}^2$. *The call* **GULA**$(\mathcal{A}, T)$ *terminates and* **GULA**$(\mathcal{A}, T) = P_{\mathcal{O}}(T)$.

Making the algorithm semantic-free atleast for those three semantics.

Theorem (Semantic-freeness)

Let P *be a* $\mathcal{M}\mathrm{VLP}$ *such that* P *is complete.*

- **GULA**$(\mathcal{A}, \mathcal{T}_{syn}(P)) = P_{\mathcal{O}}(\mathcal{T}_{syn}(P))$
- **GULA**$(\mathcal{A}, \mathcal{T}_{asyn}(P)) = P_{\mathcal{O}}(\mathcal{T}_{asyn}(P))$
- **GULA**$(\mathcal{A}, \mathcal{T}_{gen}(P)) = P_{\mathcal{O}}(\mathcal{T}_{gen}(P))$

Victory! In theory, but how does it scale in practice?

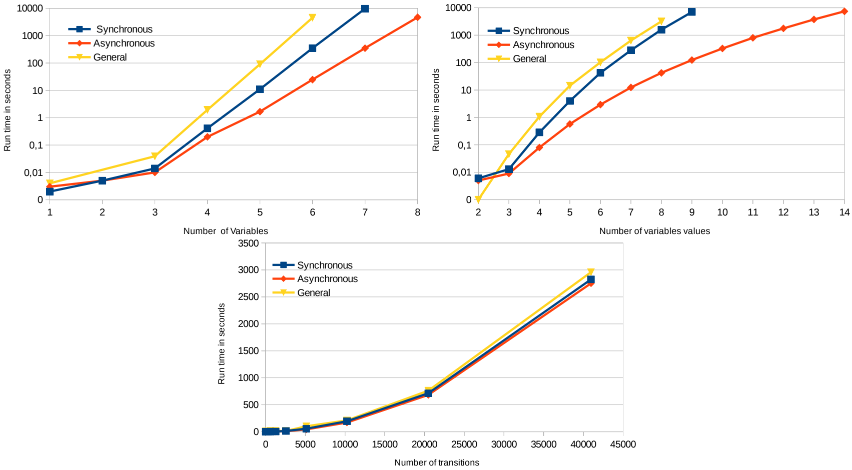
Outline

Evaluation

Theorem (**GULA** Complexity)

Let $T \subseteq \mathcal{S}^2$ be a set of transitions, $n := |\mathcal{V}|$ be the number of variables of the system and $d := \max(\text{dom}(\mathcal{V}))$ be the maximal number of values of its variables. The worst-case time complexity of **GULA** *when learning from T belongs to $\mathcal{O}(|T|^2 + 2n^3 d^{2n+1} + 2n^2 d^n)$ and its worst-case memory use belongs to $\mathcal{O}(d^{2n} + 2d^n + nd^{n+2})$.*

Evaluation



Evaluation of **GULA**'s scalability w.r.t. number of variables (top left), number of variables values (top right) and number of input transitions (bottom).

Evaluation

Semantics	Mammalian (10)	Fission (10)	Budding (12)	Arabidopsis (15)
Synchronous	1.84s / 1,024	1.55s / 1,024	34.48s / 4,096	2,066s / 32,768
Asynchronous	19.88s / 4,273	19.18s / 4,217	523s / 19,876	T.O. / 213,127
General	928s / 34,487	1,220s / 29,753	T.O. / 261,366	T.O. / $>$ 500,000

Run time of **GULA** (run time in seconds / number of transitions as input) for Boolean network benchmarks up to 15 nodes for the three semantics.

Conclusion

Conclusion

Previous works

Conclusion

Previous works

- Synchronous deterministic

Conclusion

Previous works

- Synchronous deterministic
- Markov(k) systems

Conclusion

Previous works

- Synchronous deterministic
- Markov(k) systems
- Synchronous non-deterministic (no minimality)

Conclusion

Previous works

- Synchronous deterministic
- Markov(k) systems
- Synchronous non-deterministic (no minimality)
- continuous valued systems

Conclusion

Previous works

- Synchronous deterministic
- Markov(k) systems
- Synchronous non-deterministic (no minimality)
- continuous valued systems

New contribution

Conclusion

Previous works

- Synchronous deterministic
- Markov(k) systems
- Synchronous non-deterministic (no minimality)
- continuous valued systems

New contribution

- Synchronous non-deterministic

Conclusion

Previous works

- Synchronous deterministic
- Markov(k) systems
- Synchronous non-deterministic (no minimality)
- continuous valued systems

New contribution

- Synchronous non-deterministic
- Asynchronous

Conclusion

Previous works

- Synchronous deterministic
- Markov(k) systems
- Synchronous non-deterministic (no minimality)
- continuous valued systems

New contribution

- Synchronous non-deterministic
- Asynchronous
- generalized

Conclusion

Previous works

- Synchronous deterministic
- Markov(k) systems
- Synchronous non-deterministic (no minimality)
- continuous valued systems

New contribution

- Synchronous non-deterministic
- Asynchronous
- generalized

Ongoing

Conclusion

Previous works

- Synchronous deterministic
- Markov(k) systems
- Synchronous non-deterministic (no minimality)
- continuous valued systems

New contribution

- Synchronous non-deterministic
- Asynchronous
- generalized

Ongoing

- Improve implementation + approximation

Conclusion

Previous works

- Synchronous deterministic
- Markov(k) systems
- Synchronous non-deterministic (no minimality)
- continuous valued systems

New contribution

- Synchronous non-deterministic
- Asynchronous
- generalized

Ongoing

- Improve implementation + approximation
- Apply to learn construction network

Conclusion

Previous works

- Synchronous deterministic
- Markov(k) systems
- Synchronous non-deterministic (no minimality)
- continuous valued systems

New contribution

- Synchronous non-deterministic
- Asynchronous
- generalized

Ongoing

- Improve implementation + approximation
- Apply to learn construction network
- Interface with MetaGol for learning semantics too

Conclusion

Previous works

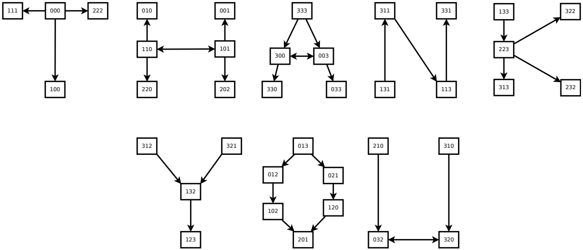
- Synchronous deterministic
- Markov(k) systems
- Synchronous non-deterministic (no minimality)
- continuous valued systems

New contribution

- Synchronous non-deterministic
- Asynchronous
- generalized

Ongoing

- Improve implementation + approximation
- Apply to learn construction network
- Interface with MetaGol for learning semantics too
- One algorithm to learn them all


111
000
222
100
010
001
110
101
220
202
333
300
003
330
033
311
331
131
113
133
322
223
313
232
312
321
132
123
013
012
021
120
102
201
210
310
032
320