A BDD-Based Algorithm for Learning from Interpretation Transition

Tony Ribeiro,
Katsumi Inoue, Chiaki Sakama

Department of Informatics
The Graduate University for Advanced Studies, Japan

National Institute of Informatics, Japan

Wakayama University, Japan

Rio de Janeiro, August 28th, 2013

Learning Dynamics of Systems

Recently, there is a growing interest in the field of Inductive Logic Programming in learning the dynamics of systems.

- Abductive action learning:
  - Abductive event calculus: Eshghi (1988), Shanahan (2000)
- Relational reinforcement learning:
  - Logic programs: Dzeroski et al. (2001)
- Learning action theories:
  - Action languages: Inoue et al.(2005), Tran & Baral (2009)
  - Probabilistic logic programs: Corapi et al. (2011)

Learning from interpretations of transitions (LFIT)

A framework for learning from interpretations of transitions (Inoue et al. 2012-2013).

- Basic Idea:
  - Learn a logic program by observing transition of interpretations of a system.
  - This logic program represents the dynamics of the system.

Example (Applications)

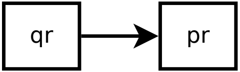
- **Bioinformatics**: Given a series of experimental observations, automatically construct a gene regulatory network.
- **Cellular Automata**: Given a sequence of transitions of the global system identify Cells local rules.
- **System design**: Given a series of desirable state transitions, construct a system that can realize those transitions.

Learning from 1-step transition (LF1T)

LF1T: an algorithm for learning from 1-step transition (Inoue et al. 2013).

Basic ideas

- Observe transition of interpretations one-by-one
- Learn logic rules from positives transitions
- Simplify rules to generalize knowledge
- Output a logic program



Transition

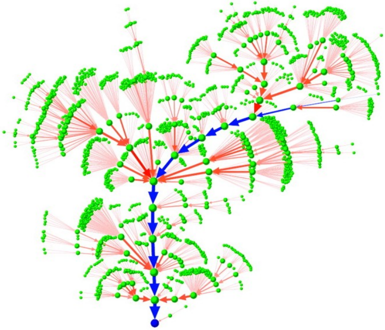
$$p \leftarrow \neg p \wedge q \wedge r.$$
$$r \leftarrow \neg p \wedge q \wedge r.$$

Normal Logic Program

Learning from 1-step transition (LF1T)

LF1T: an algorithm for learning from 1-step transition (Inoue et al. 2013).

- Properties
  - Complete
  - Sound
- Complexity
  - Memory use is exponential
  - Learning time is exponential
- Scalability
  - Input size is exponential
  - Fast learn for 16 variables

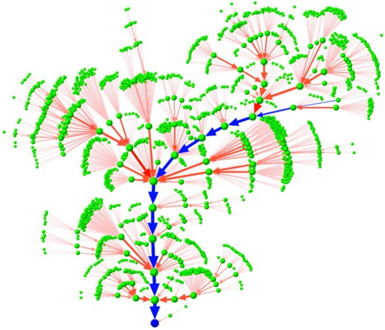


Input: State transitions (Li et al. 2004)

Learning from 1-step transition (LF1T)

LF1T: an algorithm for learning from 1-step transition (Inoue et al. 2013).

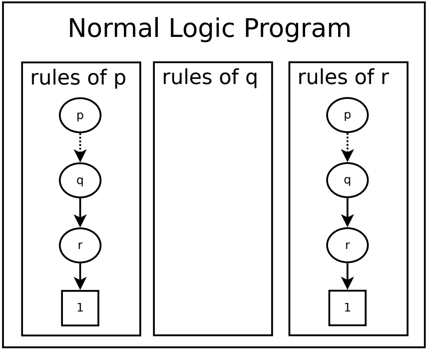
- Properties
  - Complete
  - Sound
- Complexity
  - Memory use is exponential
  - Learning time is exponential
  - Learning depends of memory
- Scalability
  - Input size is exponential
  - Fast learn for 16 variables
  - Limited to 20 variables



Input: State transitions (Li et al. 2004)

Proposal

Proposal: use Binary Decision Diagram techniques to enhance LF1T.



Binary Decision Diagrams

Interests

- Less memory
- Learn faster

Requirements

- Specific data-structure
- Dedicated operations

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

Normal Logic Program

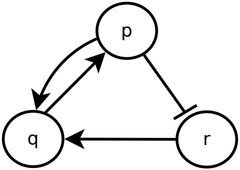
Outline

1. Preliminaries
   - Boolean Network
   - Learning From Interpretation Transitions
   - Binary Decision Diagram
2. BDD algorithm for LF1T
   - Overview
   - Operations
3. Evaluation
   - Experiments
4. Conclusion and future works

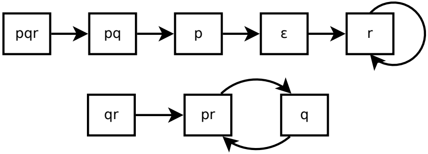
Outline

Boolean Network

A Boolean Network (Kauffman 1969), is defined by a set of variables and Boolean Functions.



Boolean Network



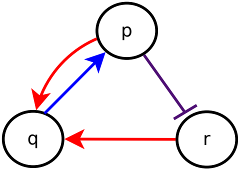
State Transitions

$$f_p = q$$
$$f_q = p \wedge r$$
$$f_r = \neg p$$

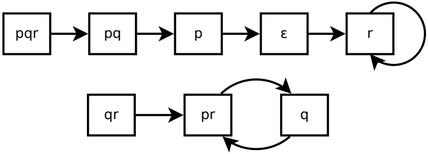
Boolean Functions

Boolean Network

Boolean Networks can be represented by Normal Logic Programs (Inoue 2011).



Boolean Network



State Transitions

$$f_p = q$$
$$f_q = p \wedge r$$
$$f_r = \neg p$$

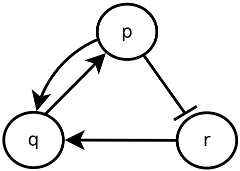
Boolean Functions

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

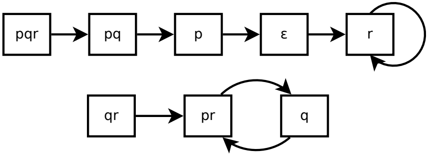
Logic Program

Boolean Network

LFIT: From the State Transitions, we can learn a Logic Program that realizes the transitions (Inoue et al. 2012-2013).



Boolean Network



State Transitions

$$f_p = q$$
$$f_q = p \wedge r$$
$$f_r = \neg p$$

Boolean Functions

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

Logic Program

Outline

Learning From Interpretation Transitions (Inoue et al. 2013)

- Herbrand interpretation I: a state of the world
- Logic program P: a state transition system, which maps an Herbrand interpretation into another interpretation (Blair et al. 1995-1997, Inoue 2011, Inoue et al. 2012)
- Next state $T_p(I)$: where T_p is the immediate consequence operator (T_p operator)
- Learning setting:
  - Given: a set of pair of Herbrand interpretations (I,J) such that $J = T_p(I)$
  - Induce a program P
- learning from interpretations (LFI)
  - Given: a set S of Herbrand interpretations
  - Induce a program P whose models are exactly S

Learning from 1 step transition (Inoue et al. 2013)

Input:

- β the Herbrand base of the system (BN's variables)
- E a set of state transitions (pair of assignment)

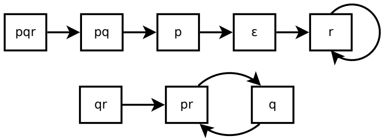
Output:

- P an NLP which represents all input state transitions

Example

$\beta = \{p, q, r\}$
$E = \{ (pqr, pq), (pq, p), (p, \epsilon), (\epsilon, r),$
$(r, r), (qr, pr)\ (pr, q)\ (q, pr)\ \}$

State transitions



Learning from 1 step transition (Inoue et al. 2013)

Input:

- β the Herbrand base of the system (BN's variables)
- E a set of state transitions (pair of assignment)

Output:

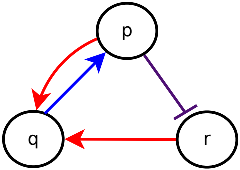
- P an NLP which represents all input state transitions

Example

$\beta = \{p, q, r\}$
$E = \{ (pqr, pq), (pq, p), (p, \epsilon), (\epsilon, r),$
$(r, r), (qr, pr)\ (pr, q)\ (q, pr)\ \}$

$P = \{\ p \leftarrow q,\ q \leftarrow p \wedge r,\ r \leftarrow \neg p\ \}$

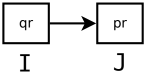
Boolean Network



Learning from 1 step transition (Inoue et al. 2013)

Let $(I, J) \in E$, for each $A \in J$ we can learn a positive rule R^I_A:

$$R^I_A := (A \leftarrow \bigwedge_{B_i \in I} B_i \wedge \bigwedge_{C_j \in \beta \setminus I} \neg C_j)$$



Example

From the state transition (qr, pr) we can learn 2 rules:

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

Generalization

Problem: output NLP is huge, almost the state transitions.

- **Goal:** a reduced NLP (#rules,#literal).
- **Why:** relevancy (size also mater in practice).
- **Idea:** use resolution to generalize the NLP.

Current NLP

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

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

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

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

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

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

$\ldots$

Reduced NLP

$p \leftarrow q.$

$q \leftarrow p \wedge r.$

$r \leftarrow \neg p.$

Naïve resolution

Let R_1, R_2 two rules and l a literal such that $l \in R_1$ and $\bar{l} \in R_2$.
R_1 and R_2 are complementary rules on l if $R_1 \setminus \{l\} = R_2 \setminus \{\bar{l}\}$.

Naïve resolution of R_1 and R_2 is $res(R_1, R_2) = R_1 \setminus \{l\}$.

Example

$R_1 := p \leftarrow p \wedge q \wedge r.$
$R_2 := p \leftarrow p \wedge q \wedge \neg r.$
$res(R_1, R_2) := p \leftarrow p \wedge q.$

Ground resolution

Let R_1, R_2 two rules and l a literal such that $l \in R_1$ and $\bar{l} \in R_2$.
R_1 can be generalised on l by R_2 if $R_2 \setminus \{\bar{l}\} \supseteq R_1 \setminus \{l\}$.

Ground resolution of R_1 by R_2 is $res(R_1, R_2) = R_1 \setminus \{l\}$.

Example

$R_1 := p \leftarrow p \wedge q \wedge r.$
$R_2 := p \leftarrow q \wedge \neg r.$

$R_2 := p \leftarrow (p \vee \neg p) \wedge q \wedge \neg r.$
$p \leftarrow p \wedge q \wedge \neg r. \supset R_2$ (complementary of R_1 on r)

$res(R_1, R_2) := p \leftarrow p \wedge q.$

Operation

LF1T analyzes each transition, one by one to learn logic rules.
Each time it learns a new rule it performs the following operations:

- Search for subsumptions
  - Is the rule subsumed by an existing one ? (Ignore)
  - Remove subsumed rules
- Search for generalizations
  - Generalize the rule (Restart)
  - Generalize the NLP by the rule
- Perform the insertion
  - Just add the rule into the NLP

Outline

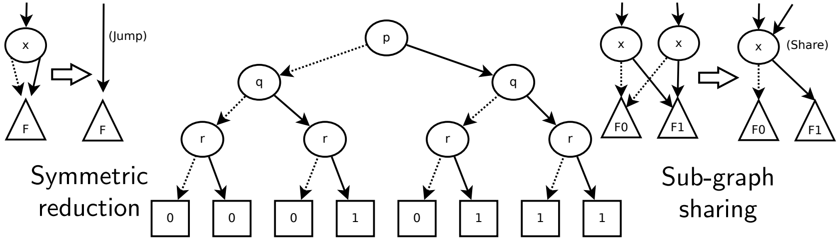
1. Preliminaries
   - Boolean Network
   - Learning From Interpretation Transitions
   - Binary Decision Diagram
2. BDD algorithm for LF1T
   - Overview
   - Operations
3. Evaluation
   - Experiments
4. Conclusion and future works

Binary Decision Diagram (BDD)

A BDD is a canonical representation of a Boolean formula (Akers 1978, Bryant 1986).

Example

$$(p \wedge q) \vee (p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge q \wedge r)$$



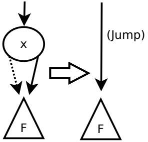
Binary Decision Diagram

Binary Decision Diagram (BDD)

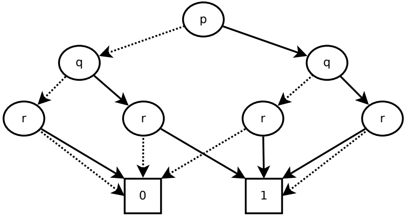
A BDD is a canonical representation of a Boolean formula (Akers 1978, Bryant 1986).

Example

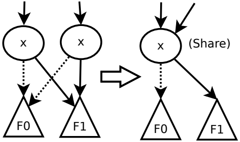
$$(p \wedge q) \vee (p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge q \wedge r)$$



Symmetric reduction



Binary Decision Diagram



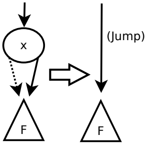
Sub-graph sharing

Binary Decision Diagram (BDD)

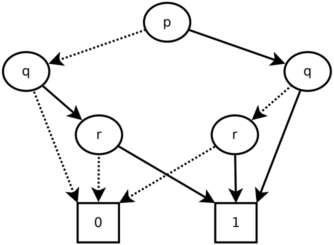
A BDD is a canonical representation of a Boolean formula (Akers 1978, Bryant 1986).

Example

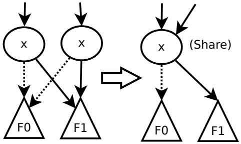
$$(p \land q) \lor (p \land \neg q \land \neg r) \lor (\neg p \land q \land r)$$



Symmetric reduction



Binary Decision Diagram



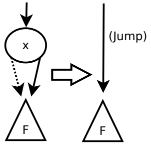
Sub-graph sharing

Binary Decision Diagram (BDD)

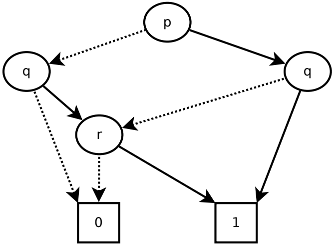
A BDD is a canonical representation of a Boolean formula (Akers 1978, Bryant 1986).

Example

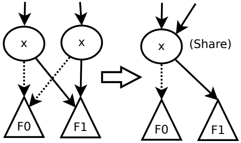
$$(p \wedge q) \vee (p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge q \wedge r)$$



Symmetric reduction



Binary Decision Diagram



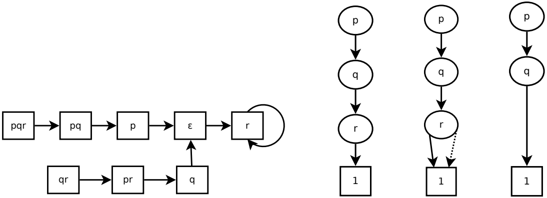
Sub-graph sharing

Outline

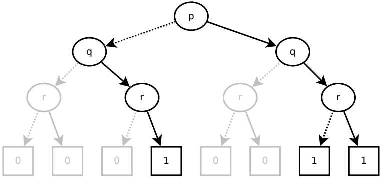
BDD algorithm for LF1T

Basic Idea

- Construct BDDs iteratively from interpretation transitions



- Learn only positive transitions: no negative leaf in our BDDs



BDD algorithm for LF1T

BDD for LF1T

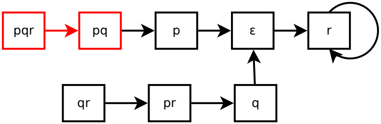
- Sub-graph sharing allows to reduce memory space
- Naïve resolution can be done by symmetric reduction

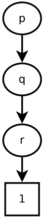
BDD algorithm for LF1T

BDD for LF1T

- Sub-graph sharing allows to reduce memory space
- Naïve resolution can be done by symmetric reduction

Transition $pqr \rightarrow pq$, learn $p \wedge q \wedge r$





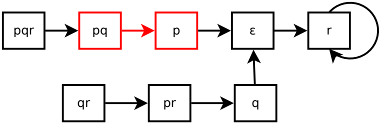
BDD of p

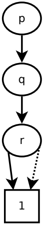
BDD algorithm for LF1T

BDD for LF1T

- Sub-graph sharing allows to reduce memory space
- Naïve resolution can be done by symmetric reduction

Transition $pq \rightarrow p$, learn $p \wedge q \wedge \neg r$





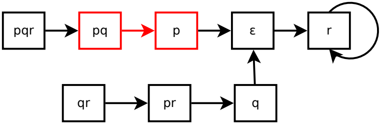
BDD of p

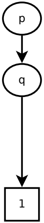
BDD algorithm for LF1T

BDD for LF1T

- Sub-graph sharing allows to reduce memory space
- Naïve resolution can be done by symmetric reduction

Naïve resolution on r, (Symmetric reduction)





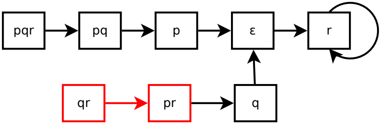
BDD of p

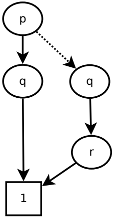
BDD algorithm for LF1T

BDD for LF1T

- Sub-graph sharing allows to reduce memory space
- Naïve resolution can be done by symmetric reduction
- But for Ground resolution we need specific methods

Transition $qr \rightarrow pr$, learn $\neg p \wedge q \wedge r$





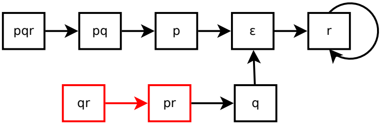
BDD of p

BDD algorithm for LF1T

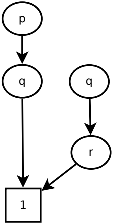
BDD for LF1T

- Sub-graph sharing allows to reduce memory space
- Naïve resolution can be done by symmetric reduction
- But for Ground resolution we need specific methods

$\neg p \wedge q \wedge r$ gives $q \wedge r$ by Ground resolution with $p \wedge q$



No existing BDD operation allows to detect this generalization.

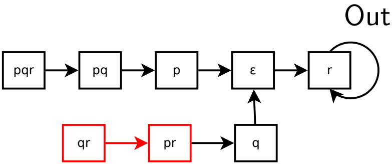


BDD of p

BDD algorithm for LF1T

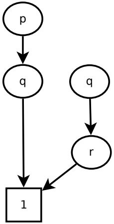
BDD for LF1T

- Sub-graph sharing allows to reduce memory space
- Naïve resolution can be done by symmetric reduction
- But for Ground resolution we need specific methods



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

$p \leftarrow p \wedge q.$
$p \leftarrow q \wedge r.$
LF1T



BDD of p

Outline

1. Preliminaries
   - Boolean Network
   - Learning From Interpretation Transitions
   - Binary Decision Diagram
2. BDD algorithm for LF1T
   - Overview
   - Operations
3. Evaluation
   - Experiments
4. Conclusion and future works

Operations

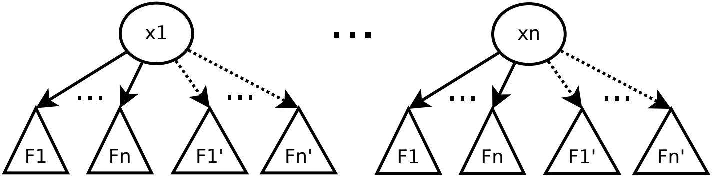
Each time we learn a new rule we have to:

- Search for subsumptions
  - Is the rule subsumed by the corresponding BDD ? (Ignore)
  - Remove subsumed parts of this BDD
- Search for generalizations
  - Generalize the rule by the BDD (Restart)
  - Generalize the BDD by the rule
- Perform the insertion
  - Ensure sub-graph sharing

Subsumption

We learn $R := h(R) \leftarrow b(R)$, and x is the first element of $b(R)$.

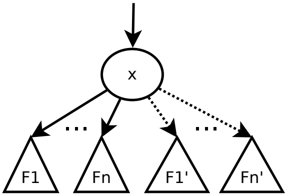
- Explore the BDD of $h(R)$ from each roots $var(root) \geq x$



Subsumption

We learn $R := h(R) \leftarrow b(R)$, and x is the first element of $b(R)$.

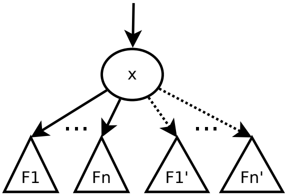
- Explore the BDD of $h(R)$ from each roots $var(root) \geq x$
- If $var(node) > x$, take the next element of $b(R)$ as x



Subsumption

We learn $R := h(R) \leftarrow b(R)$, and x is the first element of $b(R)$.

- Explore the BDD of $h(R)$ from each roots $var(root) \geq x$
- If $var(node) > x$, take the next element of $b(R)$ as x
- If $var(node) = x$, we explore corresponding sub-BDDs



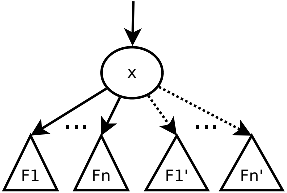
Subsumption

We learn $R := h(R) \leftarrow b(R)$, and x is the first element of $b(R)$.

- Explore the BDD of $h(R)$ from each roots $var(root) \geq x$
- If $var(node) > x$, take the next element of $b(R)$ as x
- If $var(node) = x$, we explore corresponding sub-BDDs

Termination

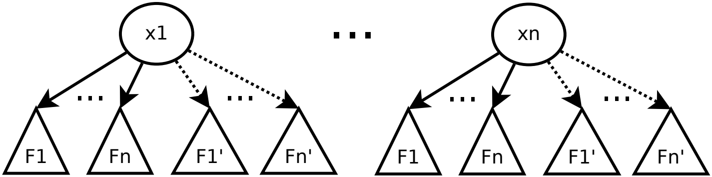
- If we reach the leaf, R is subsumed by the BDD
- If we reach the end of $b(R)$, R is not subsumed



Clean the BDD

We learn $R := h(R) \leftarrow b(R)$, and x is the first element of $b(R)$.

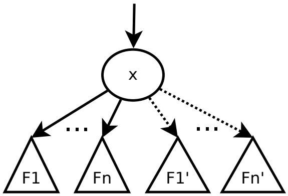
- Explore the BDD of $h(R)$ from each roots $var(root) \leq x$



Clean the BDD

We learn $R := h(R) \leftarrow b(R)$, and x is the first element of $b(R)$.

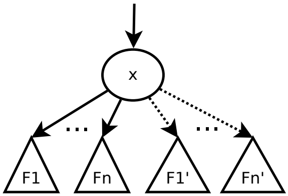
- Explore the BDD of $h(R)$ from each roots $var(root) \leq x$
- If $var(node) < x$, we explore all sub-BDDs



Clean the BDD

We learn $R := h(R) \leftarrow b(R)$, and x is the first element of $b(R)$.

- Explore the BDD of $h(R)$ from each roots $var(root) \leq x$
- If $var(node) < x$, we explore all sub-BDDs
- If $var(node) = x$, we explore corresponding sub-BDDs and take the next element of $b(R)$ as x



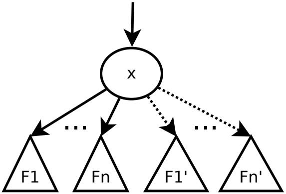
Clean the BDD

We learn $R := h(R) \leftarrow b(R)$, and x is the first element of $b(R)$.

- Explore the BDD of $h(R)$ from each roots $var(root) \leq x$
- If $var(node) < x$, we explore all sub-BDDs
- If $var(node) = x$, we explore corresponding sub-BDDs and take the next element of $b(R)$ as x

Termination

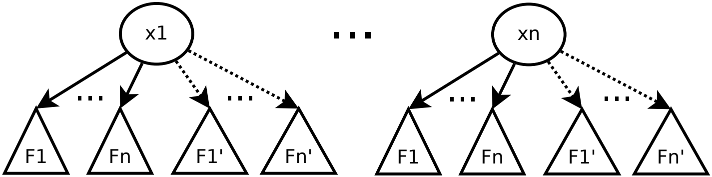
- If we reach the leaf, R cannot subsumed any more rules
- If we reach the end of $b(R)$, R subsumes all sub-BDDs rules



Generalization of the rule

We learn $R := h(R) \leftarrow b(R)$, and x is the first element of $b(R)$.

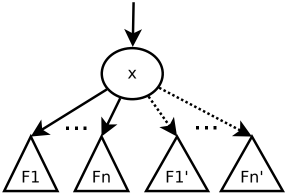
- Explore the BDD of $h(R)$ from each roots $var(root) \geq x$



Generalization of the rule

We learn $R := h(R) \leftarrow b(R)$, and x is the first element of $b(R)$.

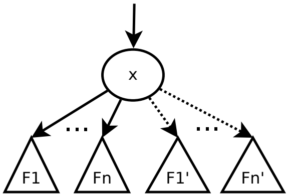
- Explore the BDD of $h(R)$ from each roots $var(root) \geq x$
- If $var(node) > x$, we take the next element of $b(R)$ as x



Generalization of the rule

We learn $R := h(R) \leftarrow b(R)$, and x is the first element of $b(R)$.

- Explore the BDD of $h(R)$ from each roots $var(root) \geq x$
- If $var(node) > x$, we take the next element of $b(R)$ as x
- If $var(node) = x$, we search a complementary rule on x: check subsumption of the rest of $b(R)$ in opposite sub-BDDs



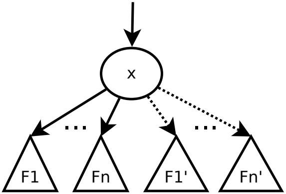
Generalization of the rule

We learn $R := h(R) \leftarrow b(R)$, and x is the first element of $b(R)$.

- Explore the BDD of $h(R)$ from each roots $var(root) \geq x$
- If $var(node) > x$, we take the next element of $b(R)$ as x
- If $var(node) = x$, we search a complementary rule on x: check subsumption of the rest of $b(R)$ in opposite sub-BDDs

Termination

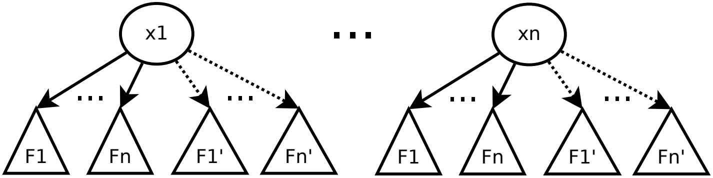
- If a sub-BDD subsumes the rest of $b(R)$, delete x from $b(R)$
- If no more elements in $b(R)$, no possible generalization



Generalization of the BDD

We learn $R := h(R) \leftarrow b(R)$, and x is the first element of $b(R)$.

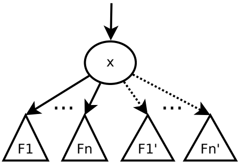
- Explore the BDD of $h(R)$ from each roots $var(root) \geq x$



Generalization of the BDD

We learn $R := h(R) \leftarrow b(R)$, and x is the first element of $b(R)$.

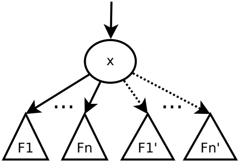
- Explore the BDD of $h(R)$ from each roots $var(root) \geq x$
- If $var(node) > x$, we take the next element of $b(R)$ as x



Generalization of the BDD

We learn $R := h(R) \leftarrow b(R)$, and x is the first element of $b(R)$.

- Explore the BDD of $h(R)$ from each roots $var(root) \geq x$
- If $var(node) > x$, we take the next element of $b(R)$ as x
- If $var(node) = x$, we extract complementary subsumed rules: rules subsumed by the rest of $b(R)$ in the opposite sub-BDDs



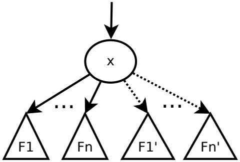
Generalization of the BDD

We learn $R := h(R) \leftarrow b(R)$, and x is the first element of $b(R)$.

- Explore the BDD of $h(R)$ from each roots $var(root) \geq x$
- If $var(node) > x$, we take the next element of $b(R)$ as x
- If $var(node) = x$, we extract complementary subsumed rules: rules subsumed by the rest of $b(R)$ in the opposite sub-BDDs

Termination

- Remove x from extracted rules
- Insert generalized rules in the BDD
- Restart the insertion of R



Insertion

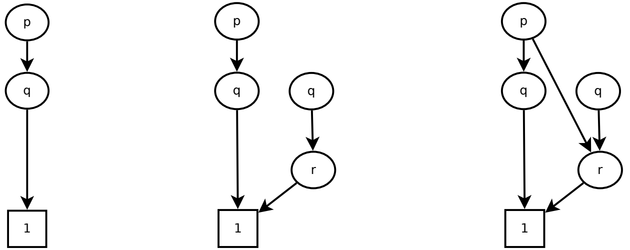
Requirements:

- R is not subsumed by the BDD
- R does not subsume any part of the BDD
- R cannot be generalized by the BDD
- R cannot generalizes the BDD

Insertion

Process:

- From the leaf node, follow the end of $b(R)$ until a node $n2$
- Select a root node where $var(root) = x$/create a new node $n1$
- Go down according to the rest of $b(R)$ until a node $n1$
- Create nodes to represent the rest of $b(R)$, connect $n1$ and $n2$



Insertion of $p \leftarrow q \wedge r$ and $p \leftarrow p \wedge r$

Outline

Settings

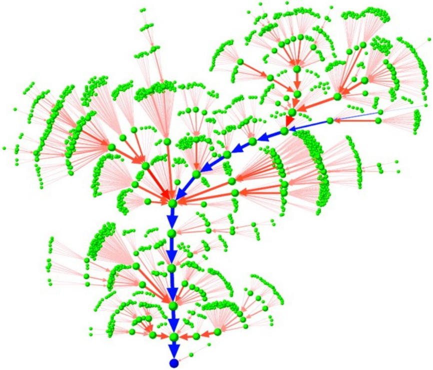
We compare our new algorithm to previous ones on some Boolean Network benchmarks from Bioinformatics literature:

Benchmarks: (Dubrova 2011)

- Mammalian cell
- Fission yeast
- Budding yeast
- Arabidopsis thalania
- Thelper

Settings:

- One run
- Time limit is 5 hours

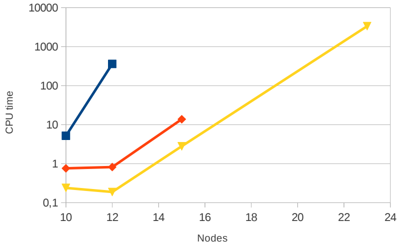


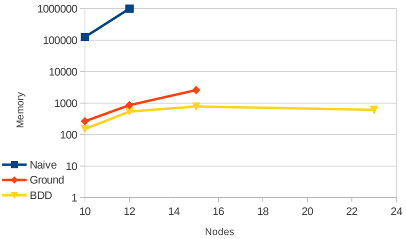
Budding yeast state transitions
(Li et al. 2004)

Results

Memory use and learning time of **LF1T** for Boolean networks up to 23 nodes with the alphabetical variable ordering

Name	# nodes	# rules	Naïve	Ground	BDD
thelper	23	26	T.O.	T.O	611/3360s
Arabidopsis	15	28	T.O.	2646/13.8s	779/2.8s
Budding	12	54	1 000 000/361s	862/0.82s	541/0.188s
Fission	10	23	126 000/5.2s	266/0.68s	147/0.24s
Mammalian	10	22	140 000/5.7s	267/0.76s	180/0.24s





Conclusion

Contribution

- A BDD algorithm to learn an NLP from state transitions.
- Outperform previous algorithms, extend the scalability.

Current and Future work

- LFkT for learning from partial state transitions
- Identification of Cellular Automata
- Density classification problem

- INOUE K., RIBEIRO T. AND SAKAMA C. 2013.
  Learning from interpretation transition.
  Machine Learning, to appear, 2013.
- INOUE K. AND SAKAMA C. 2012.
  Learning from interpretation transition.
  22nd International Conference on Inductive Logic Programming (ILP 2012), 2012.
- INOUE K. 2011.
  Logic programming for Boolean networks.
  Proceedings of the 22nd international joint conference on Artificial Intelligence (IJCAI-11), pp.924-930, AAAI Press, 2011.

Dubrova E. and Teslenko M. 2011.
A sat-based algorithm for finding attractors in synchronous boolean networks.
IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), pp.1393-1399, 2011.

Wolfram S. 2002.
A new kind of science..
Champaign: Wolfram media, 2002.

Bryant R. 1986.
Graph-based algorithms for Boolean function manipulation..
IEEE Trans. Computers, 35(8):677-691, 1986.

AKERS S. B. 1978.
Binary Decision Diagrams.
IEEE Trans. Computers, 27(6):509-516, 1978.

KAUFFMAN S. A. 1969.
Metabolic stability and epigenesis in randomly constructed genetic networks.
Journal of theoretical biology, Volume 22, Issue 3, pp.437-467, 1969.

Complexity Results

Theorem

Let n be the size of the Herbrand base $|\mathcal{B}|$. The memory complexity as well as the computational complexity of our new LF1T respectively belongs to $O(2^n)$ and $O(4^n)$.

Memory complexity

Proof.

Let n be the size of the Herbrand base $|B|$. This n is also the number of possible heads of rules and the maximum size of a rule. For each head there are 3^n possible bodies. The size of an NLP $|P|$ learned by LF1T is at most $n \cdot 3^n$.

But thanks to ground resolution, $|P|$ cannot exceed $n \cdot 2^n$; in the worst case, P contains only rules of size n where all literals appear and there is only $n \cdot 2^n$ such rules. If P contains a rule with m literals ($m < n$), this rule subsumes 2^{n-m} rules which cannot appear in P. Ground resolution also ensures that P does not contain any pair of complementary rules, so that the complexity is further divided by n; that is, $|P|$ is bounded by $O(\frac{n \cdot 2^n}{n}) = O(2^n)$. □

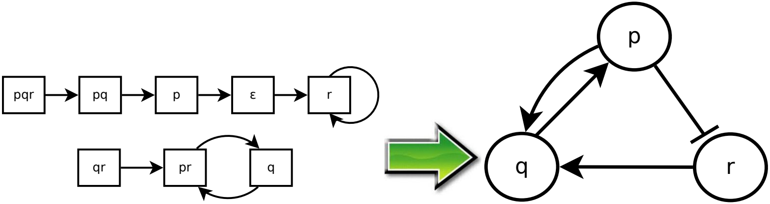
Memory complexity

Proof.

In our approach, a BDD represents all rules of P that have the same head, so that we have n BDD structures. When $|P| = 2^n$, each BDD represents $2^n/n$ rules of size n and are bound by $O(2^n/n)$, which is the upper bound size of a BDD for any Boolean function (Liaw et al. 1992). Because BDD merges common parts of rules, it is possible that a BDD that represents $2^n/n$ rules needs less than $2^n/n$ memory space. In the previous approach, in the worst case $|P| = 2^n$, whereas in our approach $|P| \leq 2^n$. Our new algorithm still remains in the same order of complexity regarding memory size: $O(2^n)$. □

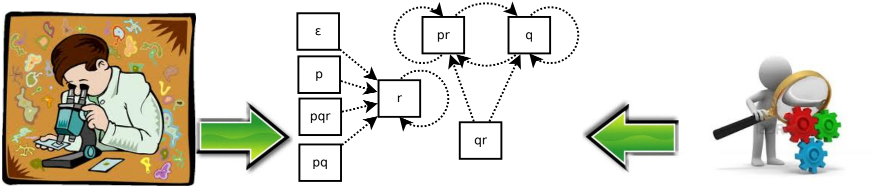
Learning from partial state transition

The weak point of LF1T is that you need to know every single transitions to learn the complete system.



Learning from partial state transition

In biology and system design, in some case we can only have the attractors or the reachability between two states.

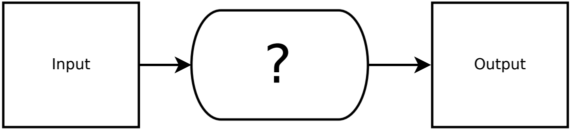


But still, we should be able to know the possible systems.

Learning from partial state transition

Idea: we can extend LF1T to learn a system without knowing the whole set of transition.

New algorithm: Given a set of input/desired output states construct a system that evolve according to it.

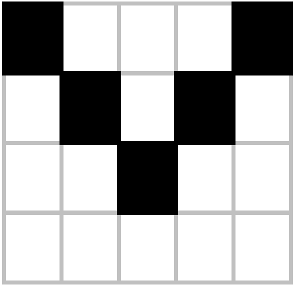


Example (Input/Output states)

- $(pq, \ldots, r)$: from pq the system evolve to the state r.
- $(r, \ldots, r)$: the state r is part of an attractor.

Application: Density Classification Task

Problem: Finding one-dimensional CA rules such that the system perform a majority vote.



Specificities:

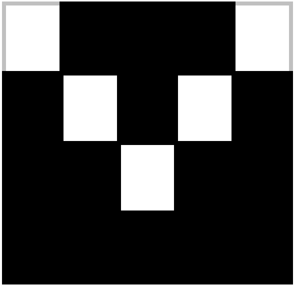
- Common rules
- Limited to neighbors

Example (Input)

{(00*,. . . ,000), (0*0,. . . ,000), (*00,. . . ,000), (11*,. . . ,111), (1*1,. . . ,111), (*11,. . . ,111)}

Application: Density Classification Task

Problem: Finding one-dimensional CA rules such that the system perform a majority vote.



Specificities:

- Common rules
- Limited to neighbors

Example (Input)

{(00*,. . . ,000), (0*0,. . . ,000), (*00,. . . ,000), (11*,. . . ,111), (1*1,. . . ,111), (*11,. . . ,111)}