The 36th IEEE International Conference on Tools with Artificial Intelligence

Linear Algebraic Partial Evaluation of Logic Programs

Tuan Nguyen[1] (speaker), Katsumi Inoue[1] and Chiaki Sakama[2]

[1] *National Institute of Informatics*, Tokyo, Japan

[2] *Wakayama University*, Wakayama, Japan

{tuannq, inoue}@nii.ac.jp sakama@wakayama-u.ac.jp

October 29^{th}, 2024

Outline

1. Motivation
2. Matrix Representation of Logic Programs
3. Linear Algebraic Partial Evaluation
   - Cycle resolving
   - Partial Evaluation with Iteration Method
   - Partial Evaluation using Matrix Decomposition
4. Experiments
5. Conclusion and future works

Outline

Motivation

- We focus on **linear algebraic charateristics of logic programs** [1].
- A **logic program** is a *set of logical rules* that can be represented in *matrices* and *vectors*.

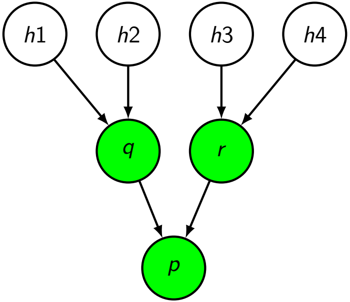
A logic program P_0:

$q \leftarrow h1 \wedge h2$,

$r \leftarrow h3 \wedge h4$,

$p \leftarrow q \wedge r$.

	$h1$	$h2$	$h3$	$h4$	p	q	r
$h1$	1						
$h2$		1					
$h3$			1				
$h4$				1			
p						1/2	1/2
q	1/2	1/2					
r			1.2	1/2			



[1] Sakama, Inoue, and Sato, "Logic programming in tensor spaces", 2021.

Motivation

Why do we need matrix representation of logic program?

- Linear algebra is at the core of many applications of scientific computation.
- Taking advantages of a long history of development in hardware/software(s) for linear algebraic computation to further *simplify the core method* and *reach higher scalability*.

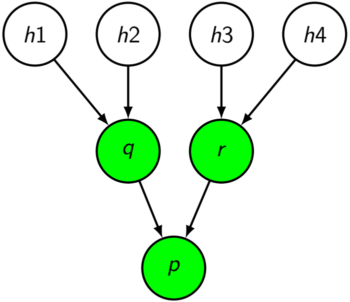
A logic program P_0:

$q \leftarrow h1 \wedge h2$,

$r \leftarrow h3 \wedge h4$,

$p \leftarrow q \wedge r$.

	$h1$	$h2$	$h3$	$h4$	p	q	r
$h1$	1						
$h2$		1					
$h3$			1				
$h4$				1			
p						1/2	1/2
q	1/2	1/2					
r			1.2	1/2			



Motivation

- **Forward reasoning** (one of the most common)

Starting with an interpretation: $\{h1, h2, h3, h4\}$:

$$
\begin{array}{c|ccccccc|}
 & h1 & h2 & h3 & h4 & p & q & r \\
\hline
h1 & 1 & & & & & & \\
h2 & & 1 & & & & & \\
h3 & & & 1 & & & & \\
h4 & & & & 1 & & & \\
p & & & & & & 1/2 & 1/2 \\
q & 1/2 & 1/2 & & & & & \\
r & & & 1.2 & 1/2 & & &
\end{array}
\cdot
\begin{array}{c|c|}
h1 & 1 \\
h2 & 1 \\
h3 & 1 \\
h4 & 1 \\
p & \\
q & \\
r &
\end{array}
=
\begin{array}{c|c|}
h1 & 1 \\
h2 & 1 \\
h3 & 1 \\
h4 & 1 \\
p & \\
q & 1 \\
r & 1
\end{array}
$$

Finish with a fixpoint: $\{h1, h2, h3, h4, p, q, r\}$:

$$
\begin{array}{c|ccccccc|}
 & h1 & h2 & h3 & h4 & p & q & r \\
\hline
h1 & 1 & & & & & & \\
h2 & & 1 & & & & & \\
h3 & & & 1 & & & & \\
h4 & & & & 1 & & & \\
p & & & & & & 1/2 & 1/2 \\
q & 1/2 & 1/2 & & & & & \\
r & & & 1.2 & 1/2 & & &
\end{array}
\cdot
\begin{array}{c|c|}
h1 & 1 \\
h2 & 1 \\
h3 & 1 \\
h4 & 1 \\
p & \\
q & 1 \\
r & 1
\end{array}
=
\begin{array}{c|c|}
h1 & 1 \\
h2 & 1 \\
h3 & 1 \\
h4 & 1 \\
p & 1 \\
q & 1 \\
r & 1
\end{array}
$$

Motivation

- **Forward reasoning** (one of the most common)

Starting with an interpretation: $\{h1, h2, h3, h4\}$:

$$\begin{array}{c} \\ h1 \\ h2 \\ h3 \\ h4 \\ p \\ q \\ r \end{array}\begin{array}{c} \begin{array}{ccccccc} h1 & h2 & h3 & h4 & p & q & r \end{array} \\ \left(\begin{array}{ccccccc} 1 & & & & & & \\ & 1 & & & & & \\ & & 1 & & & & \\ & & & 1 & & & \\ & & & & & 1/2 & 1/2 \\ 1/2 & 1/2 & & & & & \\ & & 1.2 & 1/2 & & & \end{array}\right) \end{array} \cdot \begin{array}{c} h1 \\ h2 \\ h3 \\ h4 \\ p \\ q \\ r \end{array}\left(\begin{array}{c} 1 \\ 1 \\ 1 \\ 1 \\ \\ \\ \\ \end{array}\right) = \begin{array}{c} h1 \\ h2 \\ h3 \\ h4 \\ p \\ q \\ r \end{array}\left(\begin{array}{c} 1 \\ 1 \\ 1 \\ 1 \\ \\ 1 \\ 1 \end{array}\right)$$

Finish with a fixpoint: $\{h1, h2, h3, h4, p, q, r\}$:

$$\begin{array}{c} \\ h1 \\ h2 \\ h3 \\ h4 \\ p \\ q \\ r \end{array}\begin{array}{c} \begin{array}{ccccccc} h1 & h2 & h3 & h4 & p & q & r \end{array} \\ \left(\begin{array}{ccccccc} 1 & & & & & & \\ & 1 & & & & & \\ & & 1 & & & & \\ & & & 1 & & & \\ & & & & & 1/2 & 1/2 \\ 1/2 & 1/2 & & & & & \\ & & 1.2 & 1/2 & & & \end{array}\right) \end{array} \cdot \begin{array}{c} h1 \\ h2 \\ h3 \\ h4 \\ p \\ q \\ r \end{array}\left(\begin{array}{c} 1 \\ 1 \\ 1 \\ 1 \\ \\ 1 \\ 1 \end{array}\right) = \begin{array}{c} h1 \\ h2 \\ h3 \\ h4 \\ p \\ q \\ r \end{array}\left(\begin{array}{c} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{array}\right)$$

It takes 2 steps to reach the fixpoint.

Motivation

- **Forward reasoning** (one of the most common)

Starting with an interpretation: $\{h1, h2, h3, h4\}$:

$$\begin{array}{c} \\ h1\\h2\\h3\\h4\\p\\q\\r \end{array}\begin{array}{c} \begin{array}{ccccccc} h1 & h2 & h3 & h4 & p & q & r \end{array} \\ \begin{pmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & 1 & & & & \\ & & & 1 & & & \\ & & & & & 1/2 & 1/2 \\ 1/2 & 1/2 & & & & & \\ & & 1.2 & 1/2 & & & \end{pmatrix} \end{array} \cdot \begin{array}{c} h1\\h2\\h3\\h4\\p\\q\\r \end{array}\begin{pmatrix} 1\\1\\1\\1\\ \\ \\ \end{pmatrix} = \begin{array}{c} h1\\h2\\h3\\h4\\p\\q\\r \end{array}\begin{pmatrix} 1\\1\\1\\1\\ \\1\\1 \end{pmatrix}$$

Finish with a fixpoint: $\{h1, h2, h3, h4, p, q, r\}$:

$$\begin{array}{c} \\ h1\\h2\\h3\\h4\\p\\q\\r \end{array}\begin{array}{c} \begin{array}{ccccccc} h1 & h2 & h3 & h4 & p & q & r \end{array} \\ \begin{pmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & 1 & & & & \\ & & & 1 & & & \\ & & & & & 1/2 & 1/2 \\ 1/2 & 1/2 & & & & & \\ & & 1.2 & 1/2 & & & \end{pmatrix} \end{array} \cdot \begin{array}{c} h1\\h2\\h3\\h4\\p\\q\\r \end{array}\begin{pmatrix} 1\\1\\1\\1\\ \\1\\1 \end{pmatrix} = \begin{array}{c} h1\\h2\\h3\\h4\\p\\q\\r \end{array}\begin{pmatrix} 1\\1\\1\\1\\1\\1\\1 \end{pmatrix}$$

It takes 2 steps to reach the fixpoint.

- **Backward reasoning** (similarly with a *transposed program matrix*)

Motivation

- **Partial evaluation** [2] is a technique to *simplify a logic program* by *pre-evaluating some of its parts*.

[2] Lloyd and Shepherdson, "Partial evaluation in logic programming", 1991.

Motivation

- **Partial evaluation** [2] is a technique to *simplify a logic program* by *pre-evaluating some of its parts*.

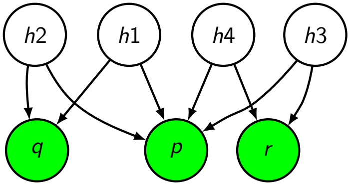
A logic program P'_0:

$q \leftarrow h1 \wedge h2$,

$r \leftarrow h3 \wedge h4$,

$p \leftarrow h1 \wedge h2 \wedge h3 \wedge h4$.

	$h1$	$h2$	$h3$	$h4$	p	q	r
$h1$	1						
$h2$		1					
$h3$			1				
$h4$				1			
p	1/4	1/4	1/4	1/4			
q	1/2	1/2					
r			1.2	1/2			



Starting from the same interpretation: $\{h1, h2, h3, h4\}$

[2] Lloyd and Shepherdson, "Partial evaluation in logic programming", 1991.

Motivation

- **Partial evaluation** [2] is a technique to *simplify a logic program* by *pre-evaluating some of its parts*.

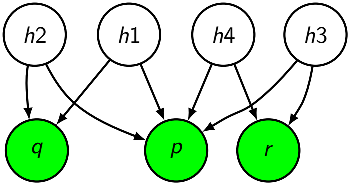
A logic program P'_0:

$q \leftarrow h1 \wedge h2$,

$r \leftarrow h3 \wedge h4$,

$p \leftarrow h1 \wedge h2 \wedge h3 \wedge h4$.

	$h1$	$h2$	$h3$	$h4$	p	q	r
$h1$	1						
$h2$		1					
$h3$			1				
$h4$				1			
p	1/4	1/4	1/4	1/4			
q	1/2	1/2					
r			1.2	1/2			



Starting from the same interpretation: $\{h1, h2, h3, h4\}$

With this program matrix, it takes only 1 steps to reach the fixpoint.

[2] Lloyd and Shepherdson, "Partial evaluation in logic programming", 1991.

Motivation

- **Partial evaluation** [2] is a technique to *simplify a logic program* by *pre-evaluating some of its parts*.

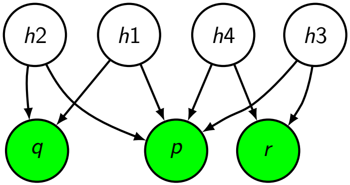
A logic program P'_0:

$q \leftarrow h1 \wedge h2$,

$r \leftarrow h3 \wedge h4$,

$p \leftarrow h1 \wedge h2 \wedge h3 \wedge h4$.

	h1	h2	h3	h4	p	q	r
h1	1						
h2		1					
h3			1				
h4				1			
p	1/4	1/4	1/4	1/4			
q	1/2	1/2					
r			1.2	1/2			



Starting from the same interpretation: $\{h1, h2, h3, h4\}$

With this program matrix, it takes only 1 steps to reach the fixpoint.

How do we transform the program matrix of P_0 into the program matrix of P'_0?

[2] Lloyd and Shepherdson, "Partial evaluation in logic programming", 1991.

Motivation

- **Linear algebraic partial evaluation** has been introduced for fixpoint computation [3] and extended to abduction [4]. The main idea is to *compute the power of a program matrix until it reaches a fixpoint*.
- Limitations:
  - only works with *And*-rules (conjunctions), and *Or*-rules (disjunctions) are **not** supported.
  - handling cycles in the program is **not** considered.
  - matrix decomposition is **not** considered in computing the power of a program matrix.

[3] H. D. Nguyen et al., "An efficient reasoning method on logic programming using partial evaluation in vector spaces", 2021.

[4] T. Q. Nguyen, Inoue, and Sakama, "Linear Algebraic Abduction with Partial Evaluation", 2023.

Outline

Matrix Representation of Logic Programs

- We consider logic programs in the form of **normal logic program**

$$h \leftarrow b_1 \ \wedge \ b_2 \ \wedge \ldots \wedge \ b_l \ \wedge \ \neg b_{l+1} \ \wedge \ldots \wedge \ \neg b_{l+k} \quad (l+k \geq l \geq 0) \tag{1}$$

- We treat a *negation* $\neg p$ as a special symbol equally to p.

Matrix Representation of Logic Programs

- We consider logic programs in the form of **normal logic program**

$$h \leftarrow b_1 \ \wedge \ b_2 \ \wedge \ldots \wedge \ b_l \ \wedge \ \neg b_{l+1} \ \wedge \ldots \wedge \ \neg b_{l+k} \tag{1}$$
$$(l + k \geq l \geq 0)$$

- We treat a *negation* $\neg p$ as a special symbol equally to p.

- Given a logic program: $P_1 =$
$\{a \leftarrow b \wedge c,\ a \leftarrow \neg h,\ a \leftarrow f,$
$b \leftarrow c \wedge d,$
$c \leftarrow a,\ c \leftarrow \neg g,\ c \leftarrow \neg d,$
$d \leftarrow e,$
$e \leftarrow d,$
$f \leftarrow a,\ f \leftarrow g,$
$g \leftarrow a,\ g \leftarrow \neg c,$
$h \leftarrow \neg a\}$

Matrix Representation of Logic Programs

- We consider logic programs in the form of **normal logic program**

$$h \leftarrow b_1 \ \wedge \ b_2 \ \wedge \ldots \wedge \ b_l \ \wedge \ \neg b_{l+1} \ \wedge \ldots \wedge \ \neg b_{l+k} \qquad (1)$$
$$(l + k \geq l \geq 0)$$

- We treat a *negation* $\neg p$ as a special symbol equally to p.

- Given a logic program: $P_1 =$
$\{a \leftarrow b \wedge c,\ a \leftarrow \neg h,\ a \leftarrow f,$
$b \leftarrow c \wedge d,$
$c \leftarrow a,\ c \leftarrow \neg g,\ c \leftarrow \neg d,$
$d \leftarrow e,$
$e \leftarrow d,$
$f \leftarrow a,\ f \leftarrow g,$
$g \leftarrow a,\ g \leftarrow \neg c,$
$h \leftarrow \neg a\}$

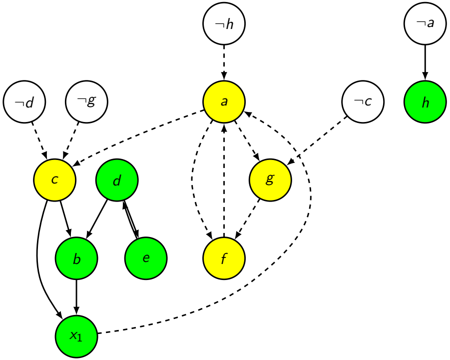
- Standardized program $\Pi_1 = \langle \Pi_1^{\wedge}, \Pi_1^{\vee}, \Pi_1^{F} \rangle$:

$\Pi_1^{\wedge} =$
$\{x_1 \leftarrow b \wedge c,$
$b \leftarrow c \wedge d,$
$h \leftarrow \neg a,$
$d \leftarrow e,$
$e \leftarrow d,\ \}$

$\Pi_1^{\vee} =$
$\{\ a \leftarrow \neg h \vee f \vee x_1,$
$c \leftarrow a \vee \neg d \vee \neg g,$
$f \leftarrow a \vee g,$
$g \leftarrow a \vee \neg c,\ \}$

$\Pi_1^{F} = \{\}$

Matrix Representation of Logic Programs



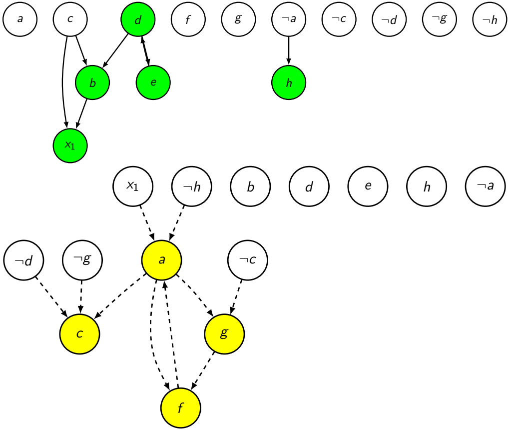
- Standardized program $\Pi_1 = \langle \Pi_1^{\wedge}, \Pi_1^{\vee}, \Pi_1^{F} \rangle$:

$\Pi_1^{\wedge} = \{x_1 \leftarrow b \wedge c,\ b \leftarrow c \wedge d,\ h \leftarrow \neg a,\ d \leftarrow e,\ e \leftarrow d,\ \}$

$\Pi_1^{\vee} = \{\ a \leftarrow \neg h \vee f \vee x_1,\ c \leftarrow a \vee \neg d \vee \neg g,\ f \leftarrow a \vee g,\ g \leftarrow a \vee \neg c,\ \}$

$\Pi_1^{F} = \{\}$

Matrix Representation of Logic Programs



- Standardized program $\Pi_1 = \langle \Pi_1^{\wedge}, \Pi_1^{\vee}, \Pi_1^{F} \rangle$:

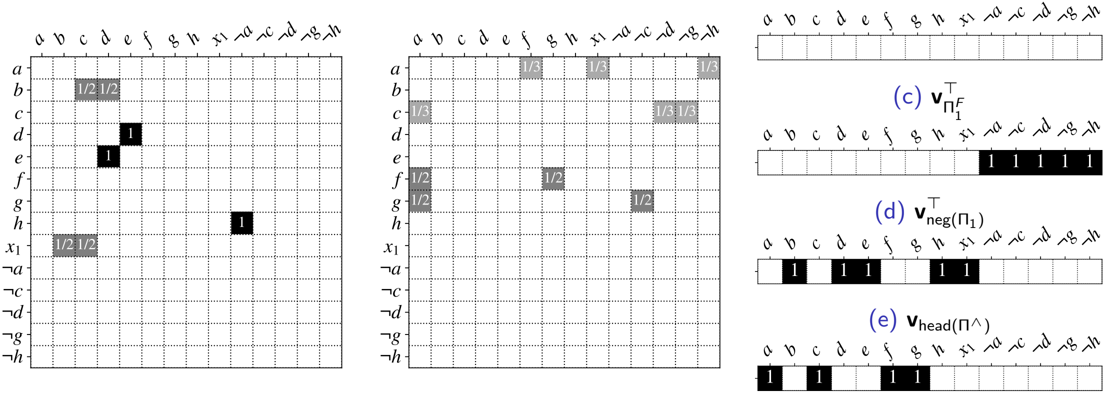
$\Pi_1^{\wedge} = \{x_1 \leftarrow b \wedge c, \; b \leftarrow c \wedge d, \; h \leftarrow \neg a, \; d \leftarrow e, \; e \leftarrow d, \}$

$\Pi_1^{\vee} = \{ a \leftarrow \neg h \vee f \vee x_1, \; c \leftarrow a \vee \neg d \vee \neg g, \; f \leftarrow a \vee g, \; g \leftarrow a \vee \neg c, \}$

$\Pi_1^{F} = \{\}$

Matrix Representation of Logic Programs

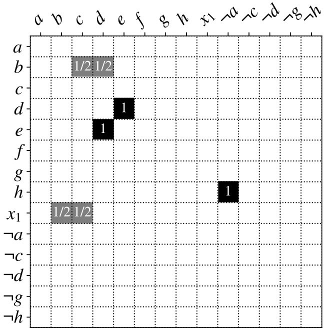
Represent Π_1 in vector spaces:



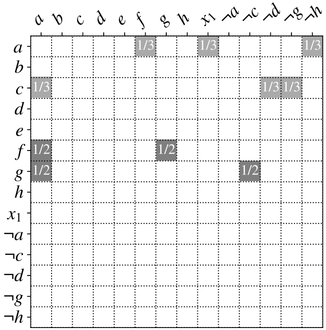
(a) $\mathbf{M}_{\Pi_1^\wedge}$ (b) $\mathbf{M}_{\Pi_1^\vee}$ (c) $\mathbf{v}^\top_{\Pi_1^F}$ (d) $\mathbf{v}^\top_{\mathrm{neg}(\Pi_1)}$ (e) $\mathbf{v}_{\mathrm{head}(\Pi^\wedge)}$ (f) $\mathbf{v}_{\mathrm{head}(\Pi^\vee)}$

Figure: Visualization of matrix/vector representations of Π_1.

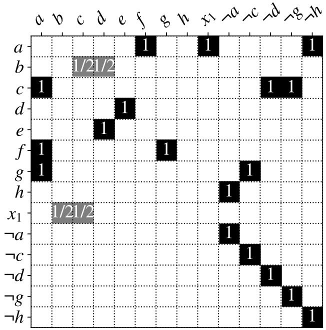
Matrix Representation of Logic Programs



(a) $\mathbf{M}_{\Pi_1^\wedge}$



(b) $\mathbf{M}_{\Pi_1^\vee}$



(c) Program matrix $\mathbf{M}_{\Pi_1}$.

Figure: The program matrix can be constructed as: $\mathbf{M}_{\Pi_1} = \mathbf{M}_{\Pi_1^\wedge} + \theta^{\Uparrow}(\mathbf{M}_{\Pi_1^\vee}) + \mathtt{diag}(\mathbf{v}_{\Pi_1^F}^\top \oplus_{\theta^\Downarrow} \mathbf{v}_{\mathrm{neg}(\Pi_1)})$.

Outline

Linear Algebraic Partial Evaluation

- **Linear algebraic Partial Evaluation (PE)** has been proposed and evaluated [5], [6], [7].
- The method is based on the iteration of *computing the matrix power*.
- It is reported to be *efficient* and *scalable* for large programs, in case we need to perform deductive/abductive reasoning for several times.

[5] Sakama, H. D. Nguyen, et al., "Partial Evaluation of Logic Programs in Vector Spaces", 2018.

[6] H. D. Nguyen et al., "An efficient reasoning method on logic programming using partial evaluation in vector spaces", 2021.

[7] T. Q. Nguyen, Inoue, and Sakama, "Linear Algebraic Abduction with Partial Evaluation", 2023.

Linear Algebraic Partial Evaluation

- **Linear algebraic PE** has been proposed and evaluated [5], [6], [7].
- The method is based on the iteration of *computing the matrix power*.
- It is reported to be *efficient* and *scalable* for large programs, in case we need to perform deductive/abductive reasoning for several times.
- Current limitations: does not consider *Or-rules*, being stuck with *cyclic programs*.

[5] Sakama, H. D. Nguyen, et al., "Partial Evaluation of Logic Programs in Vector Spaces", 2018.
[6] H. D. Nguyen et al., "An efficient reasoning method on logic programming using partial evaluation in vector spaces", 2021.
[7] T. Q. Nguyen, Inoue, and Sakama, "Linear Algebraic Abduction with Partial Evaluation", 2023.

Linear Algebraic Partial Evaluation

- **Linear algebraic PE** has been proposed and evaluated [5], [6], [7].
- The method is based on the iteration of *computing the matrix power*.
- It is reported to be *efficient* and *scalable* for large programs, in case we need to perform deductive/abductive reasoning for several times.
- Current limitations: does not consider *Or-rules*, being stuck with *cyclic programs*.
- Our proposal:
  1. Extend the method to *handle Or-rules*.
  2. *Resolve local cycles* in the program.
  3. *Employing matrix decomposition* for computing the matrix power.

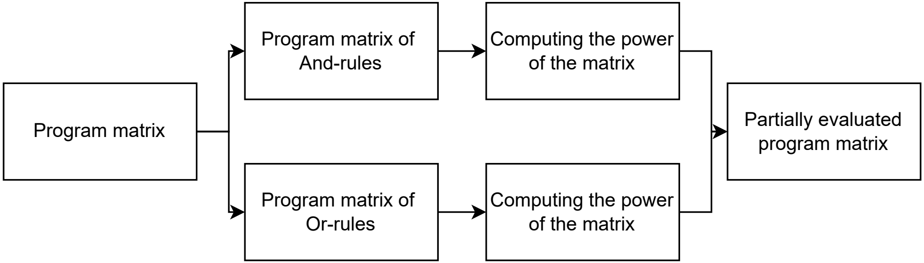
---

[5] Sakama, H. D. Nguyen, et al., "Partial Evaluation of Logic Programs in Vector Spaces", 2018.
[6] H. D. Nguyen et al., "An efficient reasoning method on logic programming using partial evaluation in vector spaces", 2021.
[7] T. Q. Nguyen, Inoue, and Sakama, "Linear Algebraic Abduction with Partial Evaluation", 2023.

Linear Algebraic Partial Evaluation

Ourproposal: Separating matrix representations of *And*-rules and *Or*-rules

- Summary of the process:



Constructing the matrix of *And*/*Or* - *2 steps*:

1. Resolve local cycles
2. Append the diagonal (to preserve information)

Computing the power of the matrix - *2 ways*:

- Iteration method
- Decomposition method

Linear Algebraic Partial Evaluation

Definition (**Partial evaluation of *And*-rules**)

Given a normal logic program P, its standardized program is Π. The partial evaluated matrix of Π w.r.t. *And*-rules, denoted as $\texttt{peval}(\Pi^{\wedge})$, is defined as follows:

$$\begin{aligned}\widehat{\mathbf{M}}_{\Pi^{\wedge}} &= \mathbf{M}_{\Pi^{\wedge}} + \texttt{diag}\big(\mathbf{v}_{\Pi^{F}} \oplus_{\theta\Downarrow} \mathbf{v}_{\text{neg}(\Pi)} \oplus_{\theta\Downarrow} \mathbf{v}_{\text{head}(\Pi^{\vee})}\big)\\ \mathbf{M}_0 &= \widehat{\mathbf{M}}_{\Pi^{\wedge}}\\ \mathbf{M}_i &= \mathbf{M}_{i-1} \cdot \mathbf{M}_{i-1} \quad (i \geq 1)\end{aligned} \tag{2}$$

where $\mathbf{v}_{\text{head}(\Pi^{\vee})}$ is a column vector such that $\mathbf{v}_{\text{head}(\Pi^{\vee})}[i] = 1$ if the corresponding atom at index i is a head of an *Or*-rule.

Linear Algebraic Partial Evaluation

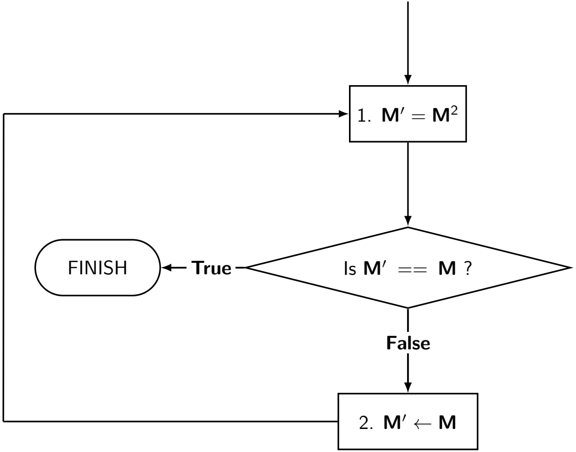
Definition (Partial evaluation of *Or*-rules)

Given a normal logic program P, its standardized program is Π. The partial evaluated matrix of Π w.r.t. *Or*-rules, denoted as $\texttt{peval}(\Pi^{\vee})$, is defined as follows:

$$
\begin{aligned}
\widehat{\mathbf{M}}_{\Pi^{\vee}} &= \mathbf{M}_{\Pi^{\vee}} + \texttt{diag}\left(\mathbf{v}_{\Pi^{F}} \oplus_{\theta\Downarrow} \mathbf{v}_{\text{neg}(\Pi)} \oplus_{\theta\Downarrow} \mathbf{v}_{\text{head}(\Pi^{\wedge})}\right) \\
\mathbf{M}_0 &= \widehat{\mathbf{M}}_{\Pi^{\vee}} \\
\mathbf{M}_i &= \mathbf{M}_{i-1} \cdot \mathbf{M}_{i-1} \quad (i \geq 1)
\end{aligned} \tag{3}
$$

where $\mathbf{v}_{\text{head}(\Pi^{\wedge})}$ is a column vector such that $\mathbf{v}_{\text{head}(\Pi^{\wedge})}[i] = 1$ if the corresponding atom at index i is a head of an *And*-rule.

Linear Algebraic Partial Evaluation



- Repeating to compute the power (2) and (3) until a fixed point is reached.

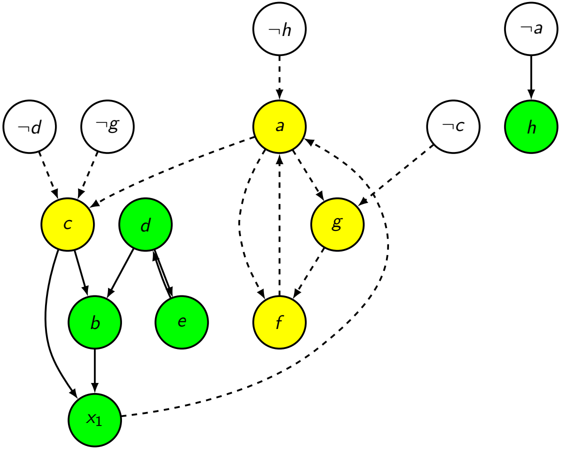
$$\mathbf{M}, \mathbf{M}^2, \mathbf{M}^4, \ldots \mathbf{M}^{2^k},$$

Proposition

For any program P with $\mathbf{M}_{\Pi^\wedge}$ (and $\mathbf{M}_{\Pi^\vee}$) of the size $n \times n$ such that the corresponding dependency graph $\mathbf{G}_{\Pi^\wedge}$ (and $\mathbf{G}_{\Pi^\wedge}$) is acyclic, the sufficient number of PE steps to reach a fixed point is $k = \lceil log_2(n) \rceil$.

Linear Algebraic Partial Evaluation - Cycle resolving

Cycle resolving: to avoid infinite loops (when a fixed point cannot be reached)



Types of cycles:

- **Local cycles**: cycles within a group of rules (only solid or only dash edges). Example: $\{d, e\}$, $\{a, f\}$, $\{a, f, g\}$.
- **Global cycles**: cycles between groups of rules (mixing both solid and dash edges). Example: $\{a, c, x_1\}$, $\{a, c, b, x_1\}$.

We focus on resolving local cycles.

Linear Algebraic Partial Evaluation - Cycle resolving

Cycle resolving: to avoid infinite loops (when a fixed point cannot be reached)

Algorithm Cycle-resolving for *And*-rules

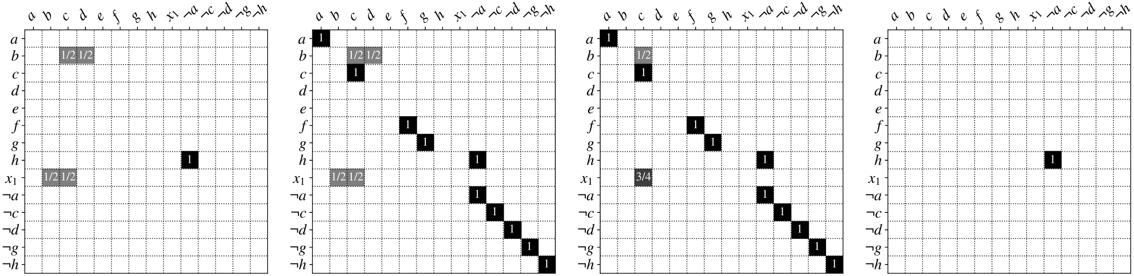
1: Identify all Strongly Connected Component (SCC)s in $\mathbf{G}_{\Pi^\wedge}$.
2: **for each** SCC L in $\mathbf{G}_{\Pi^\wedge}$ **do**
3: **for each** rule $r \in \Pi^\wedge$ such that $head(r) \in L$ **do**
4: Remove r (by setting the corresponding entries of r in $\mathbf{M}_{\Pi^\wedge}$ to 0).

Algorithm Cycle-resolving for *Or*-rules

1: Identify all SCCs in $\mathbf{G}_{\Pi^\vee}$.
2: **for each** SCC L in $\mathbf{G}_{\Pi^\vee}$ **do**
3: Let $E = \emptyset$
4: **for each** rule $r \in \Pi^\vee$ such that $head(r) \in L$ **do**
5: $E = E \cup (body(r) \setminus L)$
6: **for each** rule $r \in \Pi^\vee$ such that $head(r) \in L$ **do**
7: Replace r by $head(r) \leftarrow \bigvee_{q \in E} q$.

Linear Algebraic Partial Evaluation - with Iteration Method

Iteration Method for *And*-rules:



(a) $resolve(\mathbf{M}_{\Pi_1^{\wedge}})$ (b) $resolve(\widehat{\mathbf{M}}_{\Pi_1^{\wedge}})$. (c) $(resolve(\widehat{\mathbf{M}}_{\Pi_1^{\wedge}}))^{2^k}$ (d) $\texttt{peval}(\mathbf{M}_{\Pi_1^{\wedge}})$

Figure: Visualization of the linear algebraic PE of $\Pi_1^{\wedge}$.

Linear Algebraic Partial Evaluation - with Iteration Method

Iteration Method for *And*-rules:

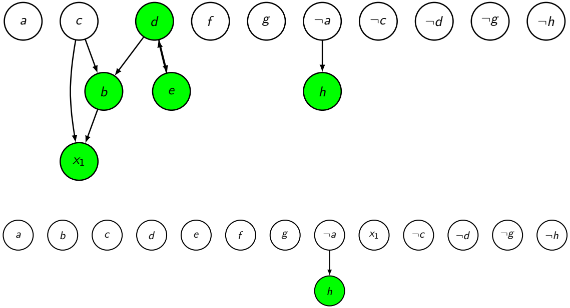
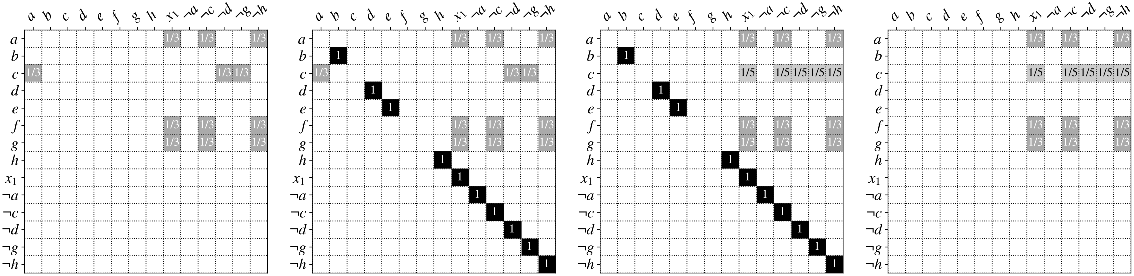


Figure: Visualization of the linear algebraic PE of $\Pi_1^{\wedge}$, before and after PE.

Linear Algebraic Partial Evaluation - with Iteration Method

Iteration Method for *Or*-rules:



(a) $resolve(\mathbf{M}_{\Pi_1^{\vee}})$ (b) $resolve(\widehat{\mathbf{M}}_{\Pi_1^{\vee}})$. (c) $(resolve(\widehat{\mathbf{M}}_{\Pi_1^{\vee}}))^{2^k}$ (d) $\mathtt{peval}(\mathbf{M}_{\Pi_1^{\vee}})$

Figure: Visualization of the linear algebraic PE of $\Pi_1^{\vee}$.

Linear Algebraic Partial Evaluation - with Iteration Method

Iteration Method for *Or*-rules:

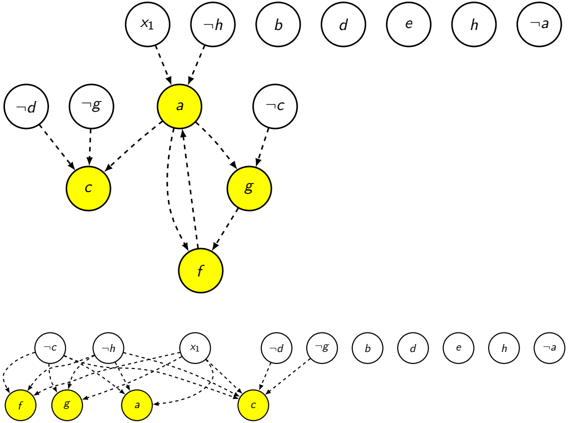
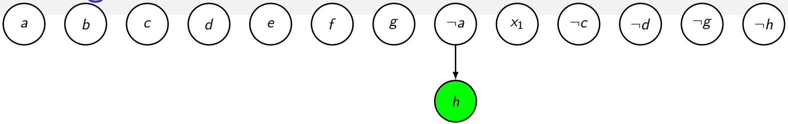
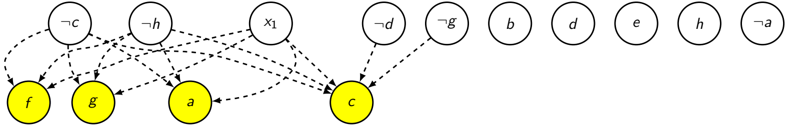


Figure: Visualization of the linear algebraic PE of $\Pi_1^{\vee}$, before and after PE.

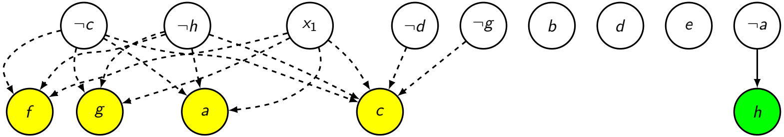
Linear Algebraic Partial Evaluation - with Iteration Method



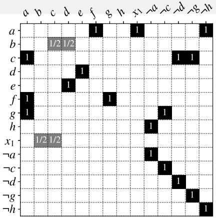
(a) Dependency graph of $\texttt{peval}(\Pi_1^{\wedge})$.



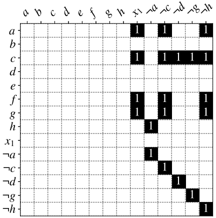
(b) Dependency graph of $\texttt{peval}(\Pi_1^{\vee})$.



(c) *And-Or*-dependency graph of $\texttt{peval}(\Pi_1)$.

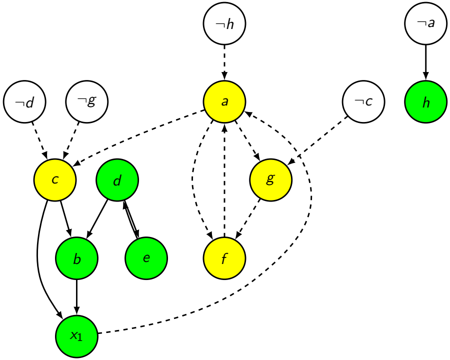


(d) $\mathbf{M}_{\Pi_1}$

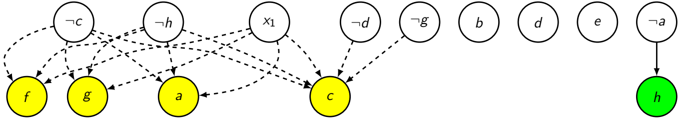


(e) $\texttt{peval}(\mathbf{M}_{\Pi_1})$

Linear Algebraic Partial Evaluation - with Iteration Method



(a) Dependency graph of $\mathbf{M}_{\Pi_1}$.



(b) Dependency graph of $\mathtt{peval}(\mathbf{M}_{\Pi_1})$.

Figure: Visualization of partial evaluated dependency graphs of Π_1 and $\mathtt{peval}(\mathbf{M}_{\Pi_1})$.

Linear Algebraic Partial Evaluation - using Matrix Decomposition

Eigendecomposition:

- It is known that powers of a matrix $\mathbf{M}$ can be computed efficiently using its decomposition $\mathbf{M} = \mathbf{Q} \cdot \mathbf{A} \cdot \mathbf{Q}^{-1}$, where $\mathbf{A}$ is a diagonal matrix of eigenvalues and $\mathbf{Q}$ is a matrix of eigenvectors [8].
- Then we can compute $\mathbf{M}^n = \mathbf{Q} \cdot \mathbf{A}^n \cdot \mathbf{Q}^{-1}$ that is computationally more efficient than computing $\mathbf{M}^n$ directly, because $\mathbf{A}$ is a diagonal matrix.

[8] Strang, *Introduction to linear algebra 4th edition*, 2009.

Linear Algebraic Partial Evaluation - using Matrix Decomposition

Eigendecomposition:

- It is known that powers of a matrix $\mathbf{M}$ can be computed efficiently using its decomposition $\mathbf{M} = \mathbf{Q} \cdot \mathbf{A} \cdot \mathbf{Q}^{-1}$, where $\mathbf{A}$ is a diagonal matrix of eigenvalues and $\mathbf{Q}$ is a matrix of eigenvectors [8].
- Then we can compute $\mathbf{M}^n = \mathbf{Q} \cdot \mathbf{A}^n \cdot \mathbf{Q}^{-1}$ that is computationally more efficient than computing $\mathbf{M}^n$ directly, because $\mathbf{A}$ is a diagonal matrix.
- Condition: the program matrix must be diagonalizable.

[8] Strang, *Introduction to linear algebra 4th edition*, 2009.

Linear Algebraic Partial Evaluation - using Matrix Decomposition

Jordan normal form:

Definition (Jordan normal form)

Let J_i be a square $k \times k$ matrix $\begin{pmatrix} \lambda_i & 1 & & & \\ & \lambda_i & 1 & & \\ & & \ddots & \ddots & \\ & & & \lambda_i & 1 \\ & & & & \lambda_i \end{pmatrix}$ such that λ_i is identical on the diagonal and there are 1s just above the diagonal. We call each such matrix a Jordan λ_i-block. A matrix $\mathbf{M}$ is in Jordan Normal Form (JNF) if $\mathbf{J} = \begin{pmatrix} J_1 & & & \\ & J_2 & & \\ & & \ddots & \\ & & & J_p \end{pmatrix}$.

- It is proved that every square matrix in $\mathbb{R}^{n \times n}$ can be decomposed into a matrix in JNF according to Jordan's theorem [9].
- $\mathbf{M} = \mathbf{P} \cdot \mathbf{J} \cdot \mathbf{P}^{-1}$

[9] Weintraub, *Jordan canonical form: theory and practice*, 2009.

Linear Algebraic Partial Evaluation - using Matrix Decomposition

Jordan normal form:

- An example of Jordan normal form:

$$\begin{pmatrix} \boxed{\begin{matrix} \lambda_1 & 1 & \\ & \lambda_1 & 1 \\ & & \lambda_1 \end{matrix}} & & & & \\ & \boxed{\begin{matrix} \lambda_2 & 1 \\ & \lambda_2 \end{matrix}} & & & \\ & & \boxed{\lambda_3} & & \\ & & & \ddots & \\ & & & & \boxed{\begin{matrix} \lambda_n & 1 \\ & \lambda_n \end{matrix}} \end{pmatrix}$$

Linear Algebraic Partial Evaluation - using Matrix Decomposition

Jordan normal form:

- Computing powers of a Jordan matrix **J** is straightforward:

$$\mathbf{J}^n = \begin{pmatrix} J_1 & & & \\ & J_2 & & \\ & & \ddots & \\ & & & J_p \end{pmatrix}^n = \begin{pmatrix} (J_1)^n & & & \\ & (J_2)^n & & \\ & & \ddots & \\ & & & (J_p)^n \end{pmatrix}$$

that can be simplified to computing powers of each Jordan block. The power of a Jordan block J_i $(k \times k)$ can be computed by:

$$(J_i)^n = \begin{pmatrix} \lambda_i^n & \binom{n}{1}\lambda_i^{n-1} & \binom{n}{2}\lambda_i^{n-2} & \cdots & \cdots & \binom{n}{k-1}\lambda_i^{n-k+1} \\ & \lambda_i^n & \binom{n}{1}\lambda_i^{n-1} & \cdots & \cdots & \binom{n}{k-2}\lambda_i^{n-k+2} \\ & & \ddots & \cdots & \cdots & \vdots \\ & & & \ddots & \cdots & \vdots \\ & & & & \lambda_i^n & \binom{n}{1}\lambda_i^{n-1} \\ & & & & & \lambda_i^n \end{pmatrix}$$

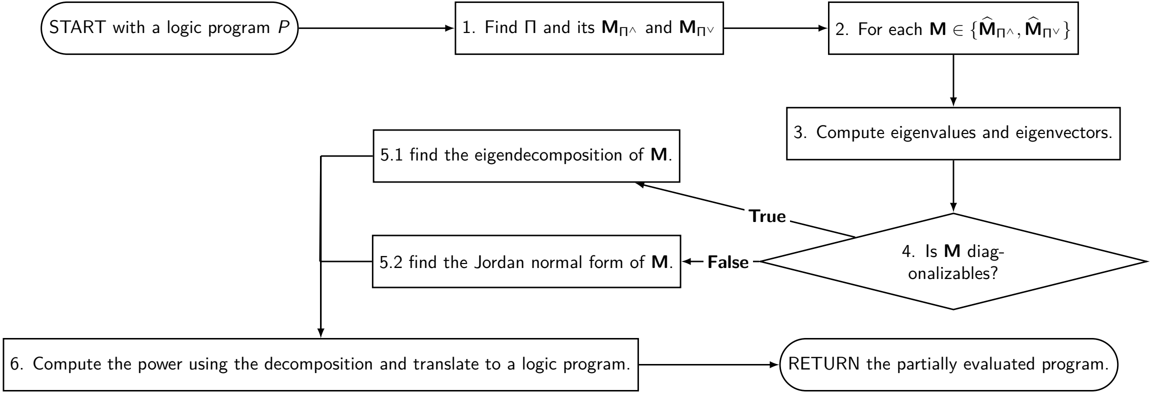
where $\binom{n}{b}$ is the binomial coefficient describing the algebraic expansion of powers of a binomial.

Linear Algebraic Partial Evaluation - using Matrix Decomposition

Algorithm Partial evaluation using matrix decomposition

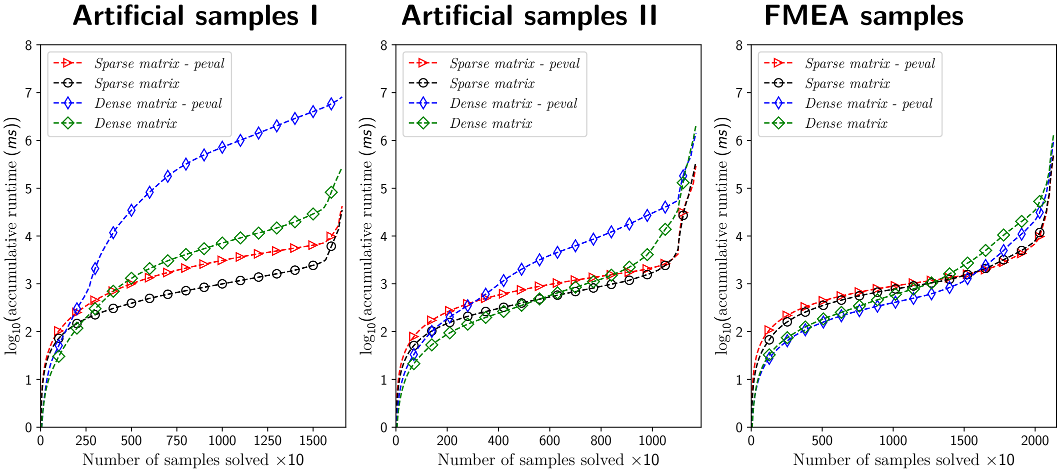
1: Find the standardized program and its matrix representation $\mathbf{M}_{\Pi^\wedge}$ and $\mathbf{M}_{\Pi^\vee}$.
2: Resolve cycles in these matrices.
3: For each matrix $\widehat{\mathbf{M}}_{\Pi^\wedge}$ and $\widehat{\mathbf{M}}_{\Pi^\vee}$, compute the eigenvalues and eigenvectors.
4: **if** the matrix is diagonalizable **then**
5: find the eigendecomposition of the matrix.
6: **else**
7: find the Jordan normal form of the matrix.
8: Compute the power using the decomposition.
9: Translate resulting matrices back to a logic program.

Linear Algebraic Partial Evaluation - using Matrix Decomposition



Outline

Experiments - (**previous work**) Propositional Horn clause abduction



PE significantly improves the performance [10]

[10] T. Q. Nguyen, Inoue, and Sakama, "Linear Algebraic Abduction with Partial Evaluation", 2023.

Experiments

- Goal:
  - evaluate linear algebraic PE with **iteration method (I)** and the **matrix decomposition method (II)** using logic programs in **Failure Modes and Effects Analysis (FMEA) benchmarks** [11].
  - evaluate performance of the methods in the *presence of cycles* in the program matrix.
- The dataset consists of three problem sets: **Artificial samples I** (166 instances), **Artificial samples II** (118 instances), and **FMEA samples** (213 instances). All programs in the dataset are *acyclic*. We *augment* the FMEA benchmarks by *adding randomly 1-5 cycles of the length 2-5 to each* $\mathbf{G}_{\Pi^\wedge}$ *and* $\mathbf{G}_{\Pi^\vee}$ of a program P.

[11] Koitz-Hristov and Wotawa, "Faster horn diagnosis-a performance comparison of abductive reasoning algorithms", 2020.

Experiments

- Our code is implemented in Python 3.7 using `numpy`, `scipy`, and `sympy`. We set a timeout of 20s for PE computation, the timeout penalty is set to 60s for comparison.
- System environment: Intel(R) Xeon(R) Bronze 3106 @1.70GHz; 64GB DDR3 @1333MHz; Ubuntu 22.04 LTS 64bit.

- All the source code and benchmark datasets in our paper will be available on GitHub:

`https://github.com/nqtuan0192/LinearAlgebraicComputationofAbduction.`

Experiments

Table: Statistical data of the datasets and detailed comparison of execution time (in *ms*) of the linear algebraic PE methods on the datasets. (green - best, red - worst)

	Artificial samples I (166 instances)		**Artificial samples II** (118 instances)		**FMEA samples** (213 instances)	
Parameters	mean / std	[min, max]	mean / std	[min, max]	mean / std	[min, max]
Matrix size	2,088.32 / 1,584.48	[11, 6,601]	321.86 / 252.64	[18, 1,110]	27.58 / 19.32	[6, 84]
No. *And*-rules	1,188.63 / 1,349.59	[8, 6,375]	201.86 / 186.64	[9, 1,007]	16.10 / 9.23	[1, 43]
No. *Or*-rules	899.69 / 839.58	[3, 3,345]	119.99 / 107.40	[4, 437]	11.48 / 11.01	[1, 41]
Sparsity (of $\mathbf{M}_{\Pi}$)	0.99 / 0.02	[0.90, 1.00]	0.99 / 0.01	[0.90, 1.00]	0.95 / 0.04	[0.73, 0.99]
Longest path	4.63 / 5.36	[2, 65]	6.56 / 8.56	[2, 58]	1.94 / 0.24	[1, 2]
`peval` steps	3.78 / 0.95	[2, 5]	3.71 / 0.81	[2, 6]	2.00 / 0.00	[2, 2]
Algorithms	mean / std	Timeout?	mean / std	Timeout?	mean / std	Timeout?
(I) Iteration + dense	799,965 / 58,500	0 / 166	4,483 / 688	0 / 118	103 / 10	0 / 213
(II) Decomposition + dense	9,292,159 / 34,274	152 / 166	6,041,323 / 28,710	96 / 118	1,607,397 / 19,170	18 / 213
(I) Iteration + sparse	545 / 15	0 / 166	138 / 4	0 / 118	157 / 5	0 / 213

Experiments

Table: Detailed comparison of execution time (in *ms*) of the linear algebraic PE methods on the *augmented* datasets **with cycles**. (green - best, red - worst)

	Artificial samples I (166 instances)		**Artificial samples II** (118 instances)		**FMEA samples** (213 instances)	
Parameters	mean / std	[min, max]	mean / std	[min, max]	mean / std	[min, max]
No. cycles *And*-rules	3.72 / 0.25	[1, 5]	3.68 / 0.30	[1, 5]	1.00 / 0.00	[1, 1]
No. cycles *Or*-rules	3.89 / 0.37	[1, 5]	3.75 / 0.42	[1, 5]	1.00 / 0.00	[1, 1]
Algorithms	`peval` (mean / std)	`resolve` (mean / std)	`peval` (mean / std)	`resolve` (mean / std)	`peval` (mean / std)	`resolve` (mean / std)
(I) Iteration + dense	821,780 / 62,340	573 / 27	4,501 / 793	407 / 19	90 / 7	52 / 6
(II) Decomposition + dense	9,251,534 / 33,491	554 / 24	5,970,126 / 27,104	398 / 18	1,271,842 / 18,510	56 / 6
(I) Iteration + sparse	579 / 17	76 / 14	151 / 4	68 / 12	112 / 4	17 / 3

Outline

Conclusion and future works

- We have proposed and investigated linear algebraic PE of logic programs:
  - extend the method to handle *Or*-rules
  - propose cycle resolving method to handle *local cycles*
  - propose decomposition method to compute the power of a program matrix

Conclusion and future works

- We have proposed and investigated linear algebraic PE of logic programs:
  - extend the method to handle *Or*-rules
  - propose cycle resolving method to handle *local cycles*
  - propose decomposition method to compute the power of a program matrix
- Future works:
  - handle *global cycles* in the program matrix.
  - investigate the effect of different rule structures on the *diagonalizability* of the program matrix.
  - explore the possibility of using *other decomposition methods*.

References I

- Koitz-Hristov, Roxane and Franz Wotawa. "Faster horn diagnosis-a performance comparison of abductive reasoning algorithms". In: *Applied Intelligence* 50.5 (2020), pp. 1558–1572.
- Lloyd, John W. and John C Shepherdson. "Partial evaluation in logic programming". In: *The Journal of Logic Programming* 11.3-4 (1991), pp. 217–242.
- Nguyen, Hien D et al. "An efficient reasoning method on logic programming using partial evaluation in vector spaces". In: *Journal of Logic and Computation* 31.5 (2021), pp. 1298–1316.
- Nguyen, Tuan Quoc, Katsumi Inoue, and Chiaki Sakama. "Linear Algebraic Abduction with Partial Evaluation". In: *Practical Aspects of Declarative Languages: 25th International Symposium, PADL 2023, Boston, MA, USA, January 16–17, 2023, Proceedings*. Springer. 2023, pp. 197–215.
- Sakama, Chiaki, Katsumi Inoue, and Taisuke Sato. "Logic programming in tensor spaces". In: *Annals of Mathematics and Artificial Intelligence* 89.12 (2021), pp. 1133–1153.
- Sakama, Chiaki, Hien D Nguyen, et al. "Partial Evaluation of Logic Programs in Vector Spaces". In: *International Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2018), Oxford UK*. 2018.

References II

- Strang, Gilbert. *Introduction to linear algebra 4th edition*. SIAM, 2009.
- Weintraub, Steven H. *Jordan canonical form: theory and practice*. Springer Nature, 2009.

Thank you for your attention

Tuan Nguyen
Email: nqtuan0192@gmail.com / tuannq@nii.ac.jp