FSM Inference from Long Traces

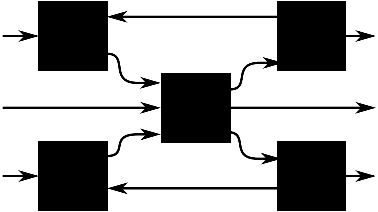
F. Avellaneda & A. Petrenko

Computer Research Institute of Montréal (Canada)

FM 2018 - 15 July 2018

Context

Increasingly modular applications

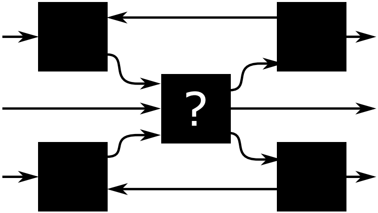


Context

Increasingly modular applications

$\Rightarrow$ no source code

$\Rightarrow$ no documentation

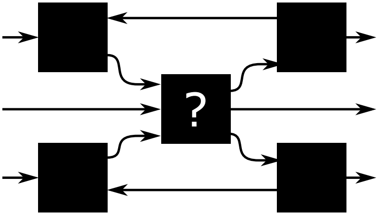


Context

Increasingly modular applications

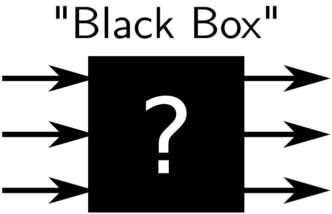
$\Rightarrow$ no source code

$\Rightarrow$ no documentation

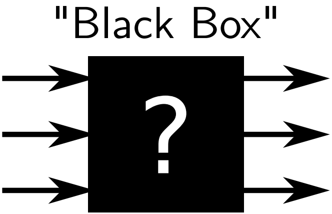


Can we trust these modules?

Problematic

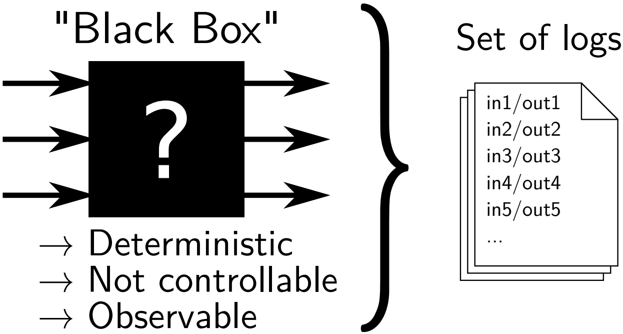


Problematic

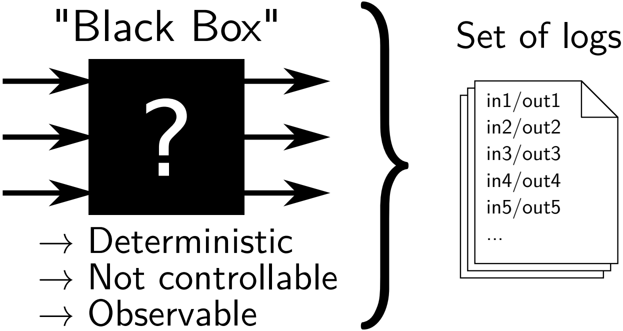


$\rightarrow$ Deterministic
$\rightarrow$ Not controllable
$\rightarrow$ Observable

Problematic



Problematic



Question: How can we infer a model for the black box from long traces?

Overview

1. Introduction
2. Passive Inference
3. Incremental Inference
4. Conclusion

Overview

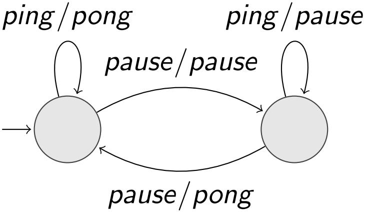
1. Introduction
2. Passive Inference
3. Incremental Inference
4. Conclusion

Model

Finite State Machine (FSM)

A FSM is a 5-tuple (S, s_0, I, O, T), where:

- S is a finite set of states with initial state s_0
- I and O are finite non-empty sets of inputs and outputs
- T is a transition relation $T \subseteq S \times I \times O \times S$



Law of parsimony: Among competing hypotheses, the one with the fewest assumptions should be selected

For FSM inference: Find a *minimal* FSM consistent with the set of traces

Formal statement of the problem

Let $\mathcal{T}$ be a set of traces and n be an integer. Find an FSM with at most n states consistent with $\mathcal{T}$

Overview

1. Introduction
2. Passive Inference
3. Incremental Inference
4. Conclusion

CSP formulation (Biermann & Feldman 1972)

The set of states Q is represented by integer variables $q_0, q_1, \ldots, q_{|Q|-1}$ taking values from 0 to $n-1$ such that:

$$\forall q_i, q_j \in Q : \text{if } q_i \not\cong q_j \text{ then } q_i \neq q_j$$
$$\text{if } \exists a \in I : \lambda(q_i, a) = \lambda(q_j, a) \text{ then}$$
$$(q_i = q_j) \Rightarrow (\Delta(q_i, a) = \Delta(q_j, a))$$

Efficient SAT formulation (Heule & Verwer 2013)

Translate a CSP formulation to SAT using unary coding for each integer variable, auxiliary variables and breaking symmetry formula

Problem: The time required to infer a model increases exponentially with the size of the logs

Overview

1 Introduction

2 Passive Inference

3 Incremental Inference

4 Conclusion

Idea: Do not consider the entire logs

First method: Use log prefixes and incrementally grow them

Algorithm 1 Infer an FSM from a set of traces

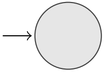
Input: A set of traces $\mathcal{T}$ and an integer n

Output: An FSM with at most n states and consistent with $\mathcal{T}$ if it exists

```
C := ∅
while C is satisfiable do
    Let M be an FSM of a solution of C
    if T ⊆ Tr_M then
        return M
    end if
    Let ω be the shortest trace in T \ Tr_M
    C := C ∧ C_ω, where C_ω has clauses encoding the fact that ω ∈ Tr_M
end while
return false
```

Example

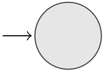
$\mathcal{T} = \mathit{ping}/\mathit{pong}.\mathit{pause}/\mathit{pause}.\mathit{ping}/\mathit{pause}.\mathit{ping}/\mathit{pause}.\mathit{pause}/\mathit{pong}$
$.\mathit{pause}/\mathit{pause}.\mathit{ping}/\mathit{pause}.\mathit{pause}/\mathit{pong}.\mathit{ping}/\mathit{pause}.\mathit{ping}/\mathit{pause}.\mathit{pause}/\mathit{pause}$
$.\mathit{ping}/\mathit{pause}.\mathit{ping}/\mathit{pause}.\mathit{pause}/\mathit{ping}.\mathit{ping}/\mathit{pong}.\mathit{ping}/\mathit{pong}\ldots$



$\omega = \epsilon$

Example

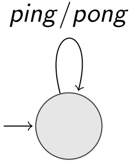
$\mathcal{T} =$ *ping / pong.pause / pause.ping / pause.ping / pause.pause / pong .pause / pause.ping / pause.pause / pong.ping / pause.ping / pause.pause / pause .ping / pause.ping / pause.pause / ping.ping / pong.ping / pong ...*



$\omega = \epsilon$

Example

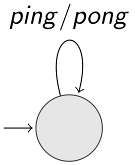
$\mathcal{T} = ping/pong.pause/pause.ping/pause.ping/pause.pause/pong$
$.pause/pause.ping/pause.pause/pong.ping/pause.ping/pause.pause/pause$
$.ping/pause.ping/pause.pause/ping.ping/pong.ping/pong\ldots$



$\omega = ping/pong$

Example

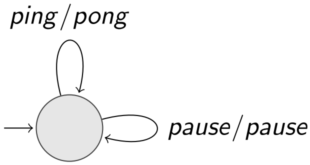
$\mathcal{T} =$ *ping / pong.pause / pause.ping / pause.ping / pause.pause / pong .pause / pause.ping / pause.pause / pong.ping / pause.ping / pause.pause / pause .ping / pause.ping / pause.pause / ping.ping / pong.ping / pong*...



$\omega =$ *ping / pong*

Example

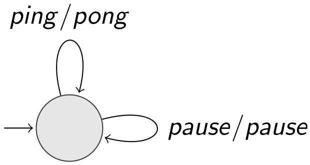
$\mathcal{T} = ping/pong.pause/pause.ping/pause.ping/pause.pause/pong$
$.pause/pause.ping/pause.pause/pong.ping/pause.ping/pause.pause/pause$
$.ping/pause.ping/pause.pause/ping.ping/pong.ping/pong\ldots$



$\omega = ping/pong.pause/pause$

Example

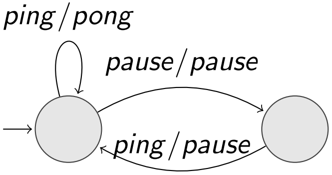
$\mathcal{T} = $ *ping / pong . pause / pause . ping / pause . ping / pause . pause / pong . pause / pause . ping / pause . pause / pong . ping / pause . ping / pause . pause / pause . ping / pause . ping / pause . pause / ping . ping / pong . ping / pong ...*



$\omega = $ *ping / pong . pause / pause*

Example

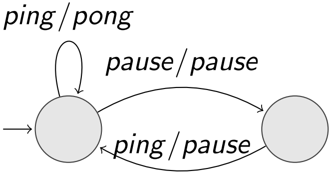
$\mathcal{T} = ping/pong.pause/pause.ping/pause.ping/pause.pause/pong$
$.pause/pause.ping/pause.pause/pong.ping/pause.ping/pause.pause/pause$
$.ping/pause.ping/pause.pause/ping.ping/pong.ping/pong\ldots$



$\omega = ping/pong.pause/pause.ping/pause$

Example

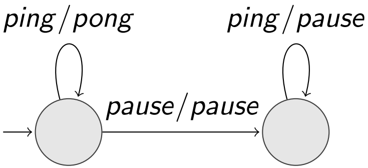
$\mathcal{T} =$ *ping / pong.pause / pause.ping / pause.ping / pause.pause / pong*
.pause / pause.ping / pause.pause / pong.ping / pause.ping / pause.pause / pause
.ping / pause.ping / pause.pause / ping.ping / pong.ping / pong...



$\omega =$ *ping / pong.pause / pause.ping / pause*

Example

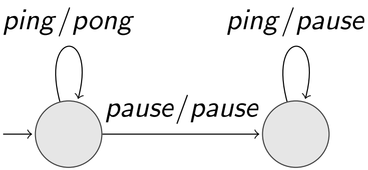
$\mathcal{T} = ping/pong.pause/pause.ping/pause.ping/pause.pause/pong$
$.pause/pause.ping/pause.pause/pong.ping/pause.ping/pause.pause/pause$
$.ping/pause.ping/pause.pause/ping.ping/pong.ping/pong\ldots$



$\omega = ping/pong.pause/pause.ping/pause.ping/pause$

Example

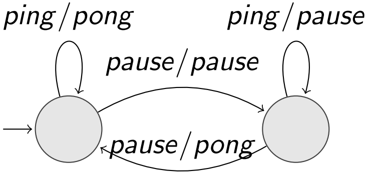
$\mathcal{T} = ping/pong.pause/pause.ping/pause.ping/pause.pause/pong$
$.pause/pause.ping/pause.pause/pong.ping/pause.ping/pause.pause/pause$
$.ping/pause.ping/pause.pause/ping.ping/pong.ping/pong\ldots$



$\omega = ping/pong.pause/pause.ping/pause.ping/pause$

Example

$\mathcal{T} = ping/pong.pause/pause.ping/pause.ping/pause.pause/pong$
$.pause/pause.ping/pause.pause/pong.ping/pause.ping/pause.pause/pause$
$.ping/pause.ping/pause.pause/ping.ping/pong.ping/pong\ldots$

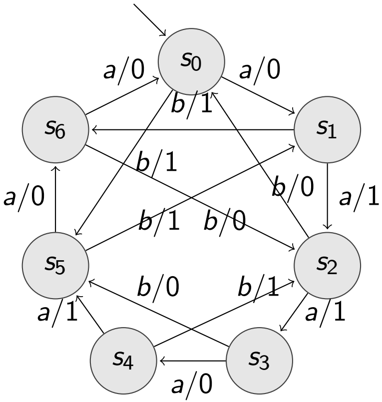


$\omega = ping/pong.pause/pause.ping/pause.ping/pause.pause/pong$

Benchmark

We calculate the average time over ten instances used to infer a random FSMs with seven states, two inputs *a* and *b*, and two outputs 0 and 1

Trace length	**H&V method (sec.)**	**Prefix method (sec.)**
1k	2.5	0.2
2k	7.3	0.2
4k	20	0.2
8k	53	0.2
16k	190	0.3
32k	Out of Memory	0.5
64k	Out of Memory	1.5



Example of a random FSM

Infrequent Events

Algorithm 2 Example client

```
while true do
    if rand() mod 1/p == 0 then
        send("pause");
    else
        send("ping");
    end if
end while
```

$\mathcal{T} = ping/pong.ping/pong.ping/pong.ping/pong.ping/pong...$
$...ping/pong.$**pause/pause**$.ping/pause.ping/pause.ping/pause...$
$...ping/pause.$**pause/pong**$.ping/pong.ping/pong.ping/pong...$

Infrequent Events

We vary the chances that *b* will appear in a 64 000 length trace produced by a randomly generated FSM

Probability of ***b***	**Prefix-based (sec.)**
50 %	1.5
25 %	1.4
10 %	1.4
1 %	3.5
0.5 %	6.0
0.3 %	16
0.2 %	90
0.1 %	Out of Memory

Infrequent Events

Idea: Add less strong constraints that refute the conjectures

Method: Let M be a conjecture and ω be the shortest trace in $\mathcal{T} \setminus Tr_M$. If there exists a suffix ω' of ω such that $\forall s : \omega' \notin Tr(s)$ then it's sufficient to add the constraint that $\exists s : \omega' \in Tr(s)$ to refute M

Observations:

- The length of ω' can be much shorter than the length of ω
- The constraint $\exists s : \omega' \in Tr(s)$ is easy to write in SAT

Algorithm 3 Infer an FSM from a set of traces

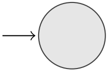
Input: A set of traces $\mathcal{T}$ and an integer n

Output: An FSM with at most n states consistent with $\mathcal{T}$ if it exists

$C := \emptyset$
while C is satisfiable **do**
    Let M be an FSM of a solution of C
    if $\mathcal{T} \subseteq Tr_M$ **then**
        return M
    end if
    Let ω be the shortest trace in $\mathcal{T} \setminus Tr_M$
    if $\exists \omega'$ the shortest suffix of ω such that $\forall s : \omega' \notin Tr(s)$ **then**
        $C := C \wedge C'_\omega$, where C'_ω has clauses encoding the fact that $\exists s : \omega' \in Tr(s)$
    else
        $C := C \wedge C_\omega$, where C_ω has clauses encoding the fact that $\omega \in Tr_M$
    end if
end while
return *false*

Example

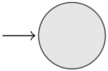
$\mathcal{T}$ = *ping / pong.ping / pong.ping / pong.ping / pong.ping / pong...*
...ping / pong.**pause/pause**.*ping / pause.ping / pause.ping / pause...*
...ping / pause.**pause/pong**.*ping / pong.ping / pong.ping / pong...*



$\omega = \epsilon$

Example

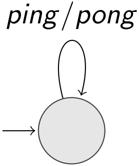
$\mathcal{T} = \mathit{ping}/\mathit{pong}.\mathit{ping}/\mathit{pong}.\mathit{ping}/\mathit{pong}.\mathit{ping}/\mathit{pong}.\mathit{ping}/\mathit{pong}\ldots$
$\ldots\mathit{ping}/\mathit{pong}.\mathbf{pause/pause}.\mathit{ping}/\mathit{pause}.\mathit{ping}/\mathit{pause}.\mathit{ping}/\mathit{pause}\ldots$
$\ldots\mathit{ping}/\mathit{pause}.\mathbf{pause/pong}.\mathit{ping}/\mathit{pong}.\mathit{ping}/\mathit{pong}.\mathit{ping}/\mathit{pong}\ldots$



$\omega = \epsilon$

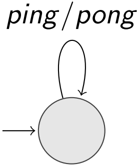
Example

$\mathcal{T}$ = *ping/pong.ping/pong.ping/pong.ping/pong.ping/pong...*
*...ping/pong.***pause/pause***.ping/pause.ping/pause.ping/pause...*
*...ping/pause.***pause/pong***.ping/pong.ping/pong.ping/pong...*



ω = *ping/pong*

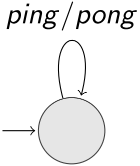
$\mathcal{T} =$ *ping/pong.ping/pong.ping/pong.ping/pong.ping/pong...*
*...ping/pong.***pause/pause***.ping/pause.ping/pause.ping/pause...*
*...ping/pause.***pause/pong***.ping/pong.ping/pong.ping/pong...*



$\omega =$ *ping/pong*

Example

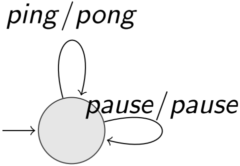
$\mathcal{T} = ping/pong.ping/pong.ping/pong.ping/pong.ping/pong...$
$...ping/pong.$**pause/pause**$.ping/pause.ping/pause.ping/pause...$
$...ping/pause.$**pause/pong**$.ping/pong.ping/pong.ping/pong...$



$\omega = ping/pong$

Example

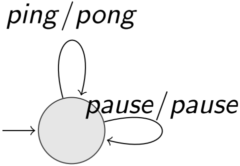
$\mathcal{T} = ping/pong.ping/pong.ping/pong.ping/pong.ping/pong...$
$...ping/pong.$**pause/pause**$.ping/pause.ping/pause.ping/pause...$
$...ping/pause.$**pause/pong**$.ping/pong.ping/pong.ping/pong...$



$$\boxed{\omega = ping/pong, \omega' = pause/pause}$$

Example

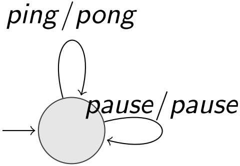
$\mathcal{T} = ping/pong.ping/pong.ping/pong.ping/pong.ping/pong \ldots$
$\ldots ping/pong.\mathbf{pause/pause}.ping/pause.ping/pause.ping/pause \ldots$
$\ldots ping/pause.\mathbf{pause/pong}.ping/pong.ping/pong.ping/pong \ldots$



$\omega = ping/pong, \omega' = pause/pause$

Example

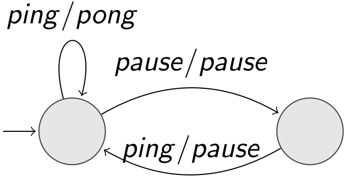
$\mathcal{T} = ping/pong.ping/pong.ping/pong.ping/pong.ping/pong...$
$...ping/pong.$**pause/pause**$.ping/pause.ping/pause.ping/pause...$
$...ping/pause.$**pause/pong**$.ping/pong.ping/pong.ping/pong...$



$$\omega = ping/pong, \omega' = pause/pause$$

Example

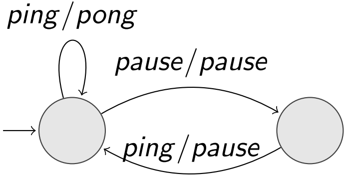
$\mathcal{T} = ping/pong.ping/pong.ping/pong.ping/pong.ping/pong...$
$...ping/pong.\mathbf{pause/pause}.ping/pause.ping/pause.ping/pause...$
$...ping/pause.\mathbf{pause/pong}.ping/pong.ping/pong.ping/pong...$



$\omega = ping/pong, \omega' = pause/pause, \omega' = ping/pause$

Example

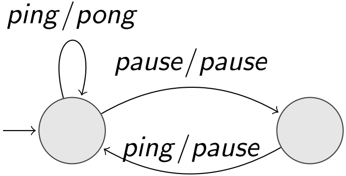
$\mathcal{T} = ping/pong.ping/pong.ping/pong.ping/pong.ping/pong...$
$...ping/pong.$**pause/pause**$.ping/pause.ping/pause.ping/pause...$
$...ping/pause.$**pause/pong**$.ping/pong.ping/pong.ping/pong...$



$\omega = ping/pong, \omega' = pause/pause, \omega' = ping/pause$

Example

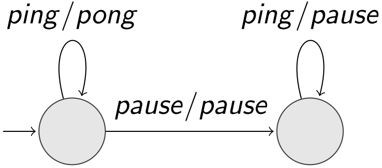
$\mathcal{T} =$ *ping/pong.ping/pong.ping/pong.ping/pong.ping/pong...*
...ping/pong.**pause/pause**.*ping/pause.ping/pause.ping/pause...*
...ping/pause.**pause/pong**.*ping/pong.ping/pong.ping/pong...*



$\omega = ping/pong, \omega' = pause/pause, \omega' = ping/pause$

Example

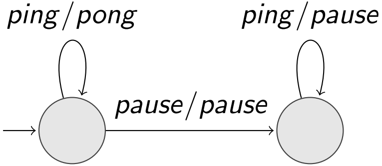
$\mathcal{T} = ping/pong.ping/pong.ping/pong.ping/pong.ping/pong...$
$...ping/pong.\textbf{pause/pause}.ping/pause.ping/pause.ping/pause...$
$...ping/pause.\textbf{pause/pong}.ping/pong.ping/pong.ping/pong...$



$\omega = ping/pong, \omega' = pause/pause, \omega' = ping/pause.ping/pause$

Example

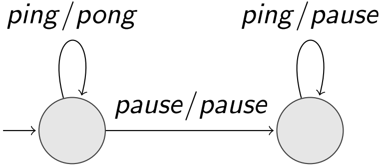
$\mathcal{T} =$ *ping / pong.ping / pong.ping / pong.ping / pong.ping / pong...*
*...ping / pong.***pause/pause***.ping / pause.ping / pause.ping / pause...*
*...ping / pause.***pause/pong***.ping / pong.ping / pong.ping / pong...*



$\omega =$ *ping / pong*, $\omega' =$ *pause / pause*, $\omega' =$ *ping / pause.ping / pause*

Example

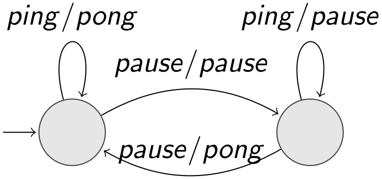
$\mathcal{T} = ping\,/\,pong.ping\,/\,pong.ping\,/\,pong.ping\,/\,pong.ping\,/\,pong...$
$...ping\,/\,pong.$**pause/pause**$.ping\,/\,pause.ping\,/\,pause.ping\,/\,pause...$
$...ping\,/\,pause.$**pause/pong**$.ping\,/\,pong.ping\,/\,pong.ping\,/\,pong...$



$\omega = ping\,/\,pong, \omega' = pause\,/\,pause, \omega' = ping\,/\,pause.ping\,/\,pause$

Example

$\mathcal{T} = ping/pong.ping/pong.ping/pong.ping/pong.ping/pong...$
$...ping/pong.$**pause/pause**$.ping/pause.ping/pause.ping/pause...$
$...ping/pause.$**pause/pong**$.ping/pong.ping/pong.ping/pong...$



$\omega = ping/pong, \omega' = pause/pause, \omega' = ping/pause.ping/pause, \omega' = pause/pong$

Benchmark

We vary the chances that *b* will appear in a 64000 length trace produced by a randomly generated FSM

Probability of ***b***	**Prefix-based (sec.)**	**Suffix-based (sec.)**
50 %	1.5	2.4
25 %	1.4	1.4
10 %	1.4	1.4
1 %	3.5	1.4
0.5 %	6.0	1.5
0.3 %	16	1.4
0.2 %	90	1.5
0.1 %	Out of Memory	1.6

Overview

1 Introduction

2 Passive Inference

3 Incremental Inference

4 Conclusion

Conclusion

Our contributions include incremental algorithms for passive FSM inference

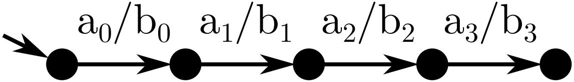
- Prefix-based algorithm
- Suffix-based algorithm

As future work, we want to investigate whether their combination will result in a more efficient method

Thank you

Table 1: Summary for encoding passive inference from ISFSM $W = (X, x_0, I, O, T)$ into SAT. n is the maximal number of states in an FSM to infer, $B = \{0, ..., n-1\}$

Clauses	**Range**
$v_{x_0,0}$	
$(v_{x,0} \vee v_{x,1} \vee ... \vee v_{x,n-1})$	$x \in X$
$(\neg v_{x,i} \vee \neg v_{x,j})$	$x \in X; 0 \leq i < j < n$
$(\neg v_{x,i} \vee \neg v_{x',i})$	$x \not\simeq x'; i \in B$
$(y_{a,i,j} \vee \neg v_{x,i} \vee \neg v_{x',j})$	$(x, a, o, x') \in T; i, j \in B$
$(\neg y_{a,i,h} \vee \neg y_{a,i,j})$	$a \in I; h, i, j \in B; h < j$
$(y_{a,i,0} \vee y_{a,i,1} \vee ... \vee y_{a,i,n-1})$	$a \in I; i \in B$
$(\neg y_{a,i,j} \vee \neg v_{x,i} \vee v_{x',j})$	$(x, a, o, x') \in T; i, j \in B$



$v_{0,0}$	$v_{1,0}$	$v_{2,0}$	$v_{3,0}$	$v_{4,0}$
$v_{0,1}$	$v_{1,1}$	$v_{2,1}$	$v_{3,1}$	$v_{4,1}$
$v_{0,2}$	$v_{1,2}$	$v_{2,2}$	$v_{3,2}$	$v_{4,2}$

Clauses	**Range**
$v_{x_0,0}$	
$(v_{x,0} \vee v_{x,1} \vee ... \vee v_{x,n-1})$	$x \in X$
$(\neg v_{x,i} \vee \neg v_{x,j})$	$x \in X; 0 \leq i < j < n$
$(\neg v_{x,i} \vee \neg v_{x',i})$	$x \not\cong x'; i \in B$
$(v_{x,i} \wedge v_{x',i}) \Rightarrow (v_{\Delta(x,a),j} \Rightarrow v_{\Delta(x',a),j})$	$x, x' \in X \mid \lambda(x,a) = \lambda(x',a); i, j \in B$