A Pattern Recognition Approach for Speculative Firing Prediction in Distributed Saturation State-Space Generation

Ming-Ying Chung and Gianfranco Ciardo

Department of Computer Science and Engineering
University of California at Riverside
Riverside, CA 92521, USA

{chung,ciardo}@cs.ucr.edu

Outline

- **Introduction**
  - *Parallel and distributed state-space generation*
- **Background**
  - *Multi-valued Decision Diagrams (MDDs)* encoding of the state-space
  - *Kronecker* encoding of the next state function
  - *Saturation-style* fixed point iteration and its distributed version
- **Speculative firing prediction**
  - Idea and the naïve approach
  - The *pattern recognition* approach
- **Experimental results**
- **Conclusions**

Introduction

- *Formal verification* : for *quality assurance*
- *Model checking* : a, model base, automatic verification approach

  E. Clarke and E. Emerson. *Synthesis of synchronization skeletons for branching time temporal logic*, Logic of Programs 1981
- *State-space generation* : the first step in model checking (memory-intensive)
- *Binary decision diagrams* (BDDs) : *symbolic* state-space construction

  R. Bryant, *Graph-based algorithm for boolean function manipulation*, IEEE TC 1986
- *Parallel and distributed model checking* :

  uses computation resources on a network or different computer architecture
- *Saturation NOW* : a *message-passing* scheme based on *saturation*

  M.-Y. Chung and G. Ciardo, *Saturation NOW*, QEST 2004

Related symbolic parallel and distributed work

- On *shared-memory multi-processor* : special hardware

  S. Kimura and E. M. Clarke, *A parallel algorithm for constructing BDDs*, ICCD 1990

- On *distributed shared memory* : special software

  Y. Parasuram, E. Stabler, and S.-K. Chin, *Parallel implementation of BDD algorithm using a DSM*, HICSS 1994

- On *network of workstations* :

  - Job-based slicing : overlap (duplicate) image computation

    T. Stornetta and F. Brewer, *An efficient parallel BDD package*, DAC 1996

    K. Milvang-Jensen and A. Hu, *BDDNOW*, FMCAD 1998

    T. Heyman, D. Geist, O. Grumberg, T. Heyman, and A. Schuster, *Achieving scalability in parallel reachability analysis of very large circuits*, CAV 2000

    O. Grumberg, T. Heyman, and A. Schuster, *A work-efficient distributed algorithm for reachability analysis*, CAV 2003

  - Level-based slicing : scaling problem, no memory load balancing

    R. Ranjan, J. Snaghavi, R. Brayton, and A. Sangiovanni-Vincentelli, *BDDs on NOWs*, ICCD 1996

Background

Structured discrete-state models

- A *structured discrete state model* is a triple $(\widehat{\mathcal{S}}, \mathcal{S}^{init}, \mathcal{N})$ where
  - $\widehat{\mathcal{S}}$ is the set of *potential states* of the model
  - $\mathcal{S}^{init} \subseteq \widehat{\mathcal{S}}$ is the set of *initial states*
  - $\mathcal{N} : \widehat{\mathcal{S}} \rightarrow 2^{\widehat{\mathcal{S}}}$ is the *next-state* function

- The *reachable state-space* $\mathcal{S} \in \widehat{\mathcal{S}}$ is the smallest set
  - containing $\mathcal{S}^{init}$
  - closed with respect to $\mathcal{N}$

$$\mathcal{S} = \mathcal{S}^{init} \cup \mathcal{N}(\mathcal{S}^{init}) \cup \mathcal{N}^2(\mathcal{S}^{init}) \cup \mathcal{N}^3(\mathcal{S}^{init}) \cup \cdots = \mathcal{N}^*(\mathcal{S}^{init})$$

Quasi-reduced ordered MDDs

- We *structure the states by level* :

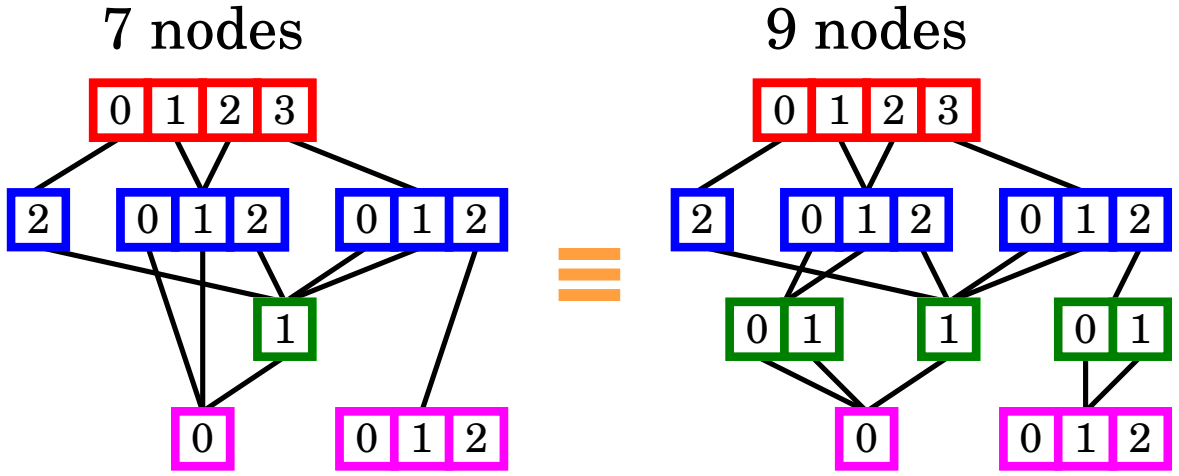
$$\mathbf{i} = (\mathbf{i}_K, \ldots, \mathbf{i}_1)$$

$\mathcal{S}_4 = \{0, 1, 2, 3\}$

$\mathcal{S}_3 = \{0, 1, 2\}$

$\mathcal{S}_2 = \{0, 1\}$

$\mathcal{S}_1 = \{0, 1, 2\}$



$$\mathcal{S} = \underbrace{\left\{ \begin{matrix} 0 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 0 & 0 & 1 & 1 & 2 & 0 & 0 & 1 & 1 & 2 & 0 & 1 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 1 & 2 \end{matrix} \right\}}_{19 \text{ states}}$$

T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli, *Multi-valued decision diagrams: theory and applications*, Journal of multiple-valued logic 1998

Quasi-reduced ordered MDDs

- We *structure the states by level* :

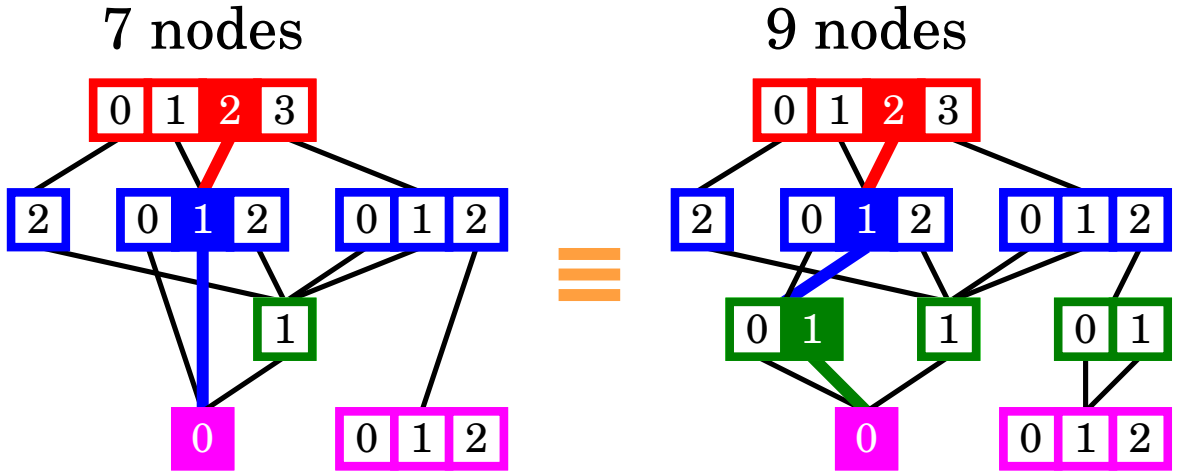
$$\mathbf{i} = (\mathbf{i}_K, \ldots, \mathbf{i}_1)$$

$\mathcal{S}_4 = \{0, 1, 2, 3\}$

$\mathcal{S}_3 = \{0, 1, 2\}$

$\mathcal{S}_2 = \{0, 1\}$

$\mathcal{S}_1 = \{0, 1, 2\}$



$$\mathcal{S} = \left\{ \begin{array}{ccccccccc|c|ccccccccc} 0 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 0 & 0 & 1 & 1 & 2 & 0 & 1 & 2 & 1 & 2 & 0 & 1 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 1 & 2 \end{array} \right\}$$

19 states

T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli, *Multi-valued decision diagrams: theory and applications*, Journal of multiple-valued logic 1998

Quasi-reduced ordered MDDs

- We *structure the states by level* :

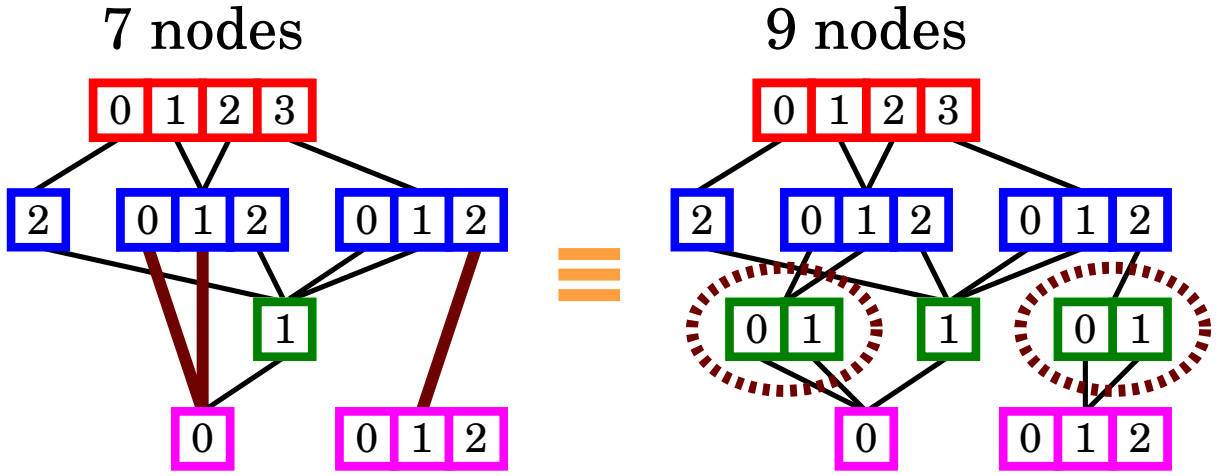
$$\mathbf{i} = (\mathbf{i}_K, \ldots, \mathbf{i}_1)$$

$\mathcal{S}_4 = \{0, 1, 2, 3\}$

$\mathcal{S}_3 = \{0, 1, 2\}$

$\mathcal{S}_2 = \{0, 1\}$

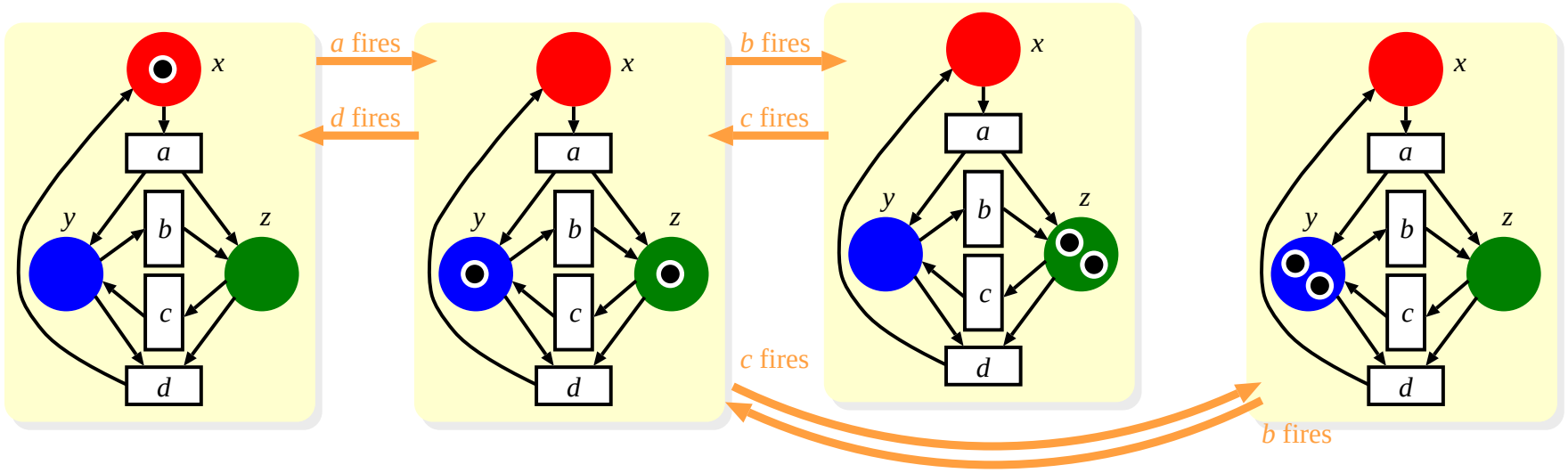
$\mathcal{S}_1 = \{0, 1, 2\}$



$$\mathcal{S} = \underbrace{\left\{ \begin{matrix} 0 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 0 & 0 & 1 & 1 & 2 & 0 & 0 & 1 & 1 & 2 & 0 & 1 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 1 & 2 \end{matrix} \right\}}_{19 \text{ states}}$$

T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli, *Multi-valued decision diagrams: theory and applications*, Journal of multiple-valued logic 1998

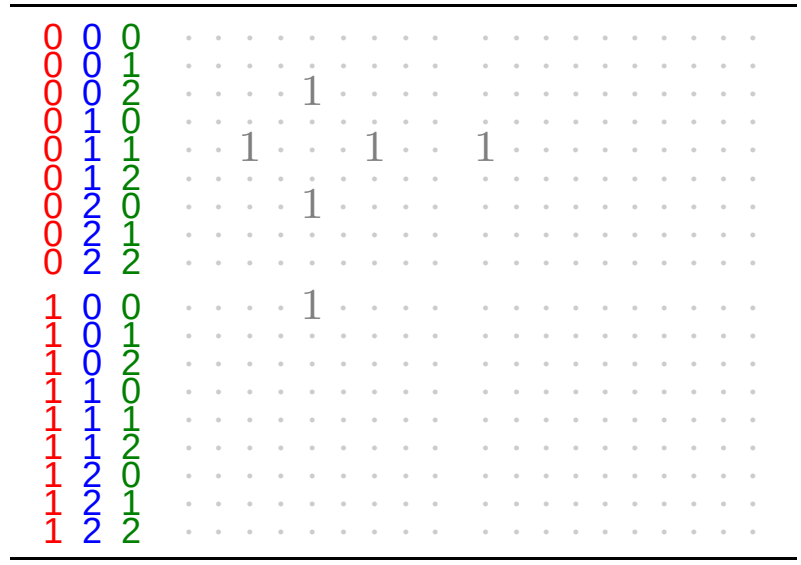
Incidence matrix for $\mathcal{N}$



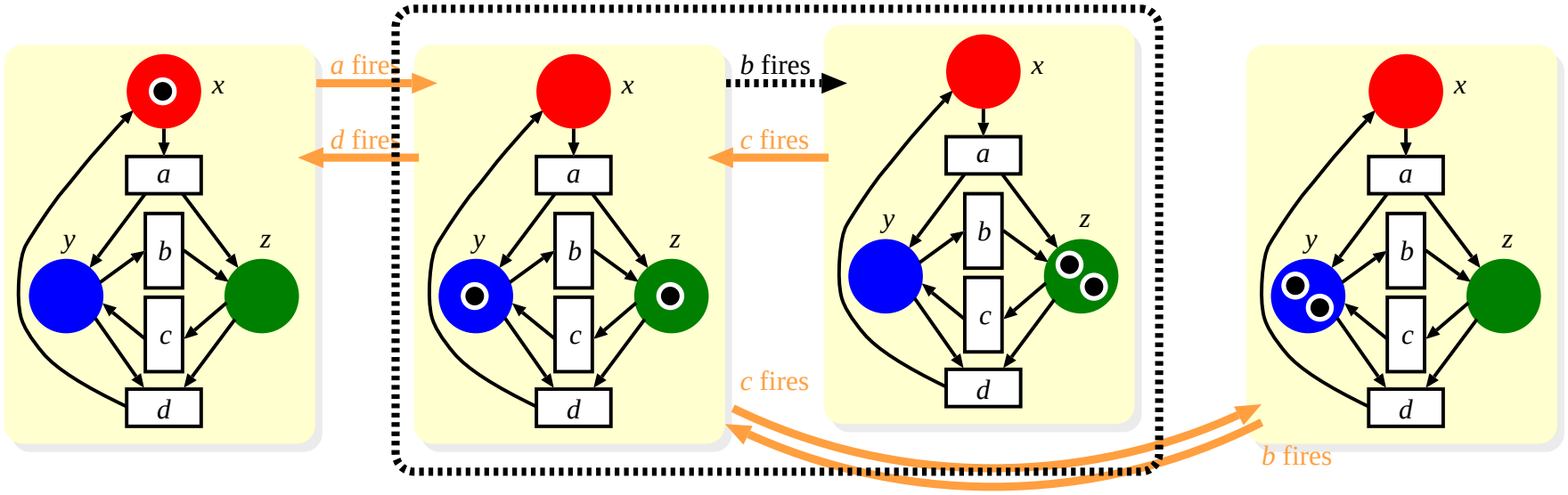
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 1 1 1 2 2 2 0 0 0 0 1 1 1 2 2 2
0 1 2 0 1 2 0 1 2 0 1 2 0 0 1 2 0 1 2

Incidence matrix for $\mathcal{N}$

$$\mathbf{N}[\mathbf{i},\mathbf{j}] = 1 \;\Leftrightarrow\; \mathbf{j} \in \mathcal{N}(\mathbf{i})$$



Incidence matrix for $\mathcal{N}$

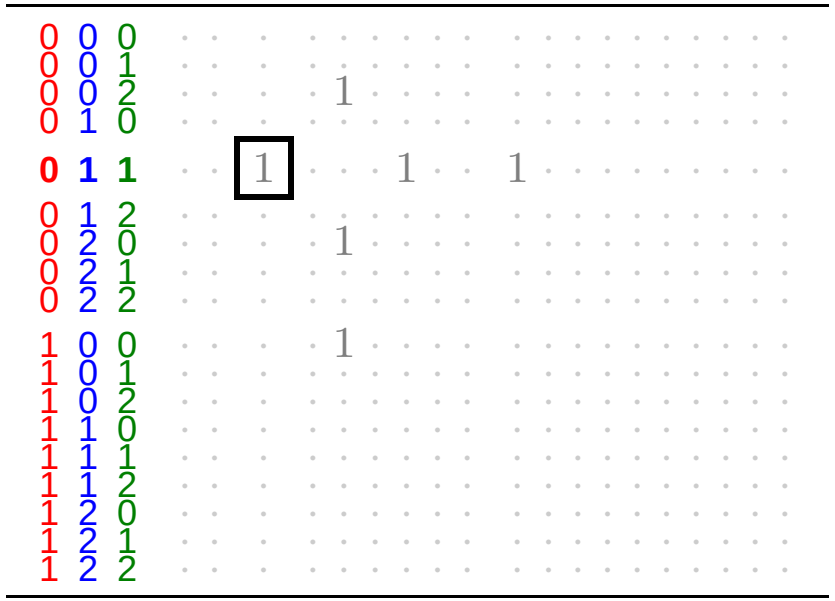


00 0 000000 111111111111
00 0 111222 000011122 2
01 2 012012 012001201 2

Incidence matrix for $\mathcal{N}$

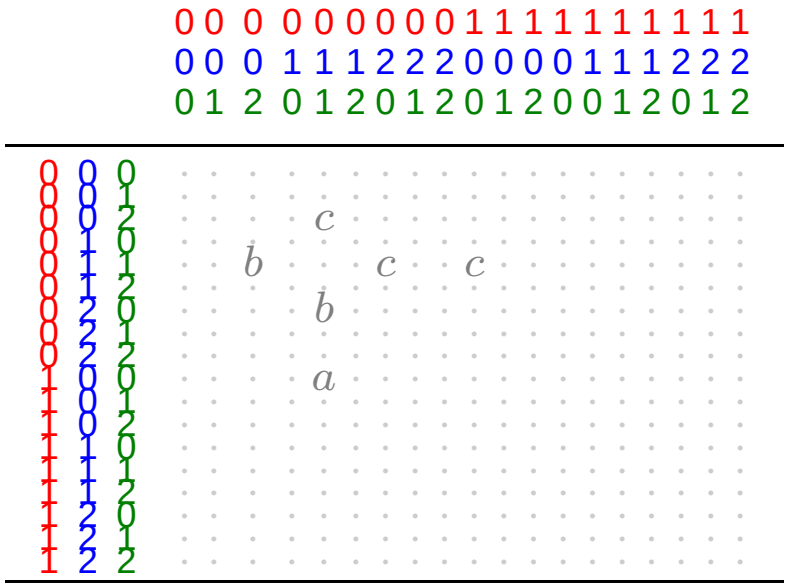
$$\mathbf{N}[\mathbf{i},\mathbf{j}] = 1 \;\Leftrightarrow\; \mathbf{j} \in \mathcal{N}(\mathbf{i})$$

$\mathbf{N}$ is extremely sparse



Kronecker encoding of the $\mathbf{N}$ $(K = 3)$

$$\mathbf{N} \;=\; \sum_{e\in\mathcal{E}} \mathbf{N}_e \;=\; \sum_{e\in\mathcal{E}} \bigotimes_{K\geq k\geq 1} \mathbf{N}_{k,e}$$



	a	b	c	d
x	$1\to 0$	$\mathbf{I}$	$\mathbf{I}$	$0\to 1$
y	$0\to 1$	$1\to 0$ $2\to 1$	$0\to 1$ $1\to 2$	$1\to 0$
z	$0\to 1$	$0\to 1$ $1\to 2$	$2\to 1$ $1\to 0$	$1\to 0$

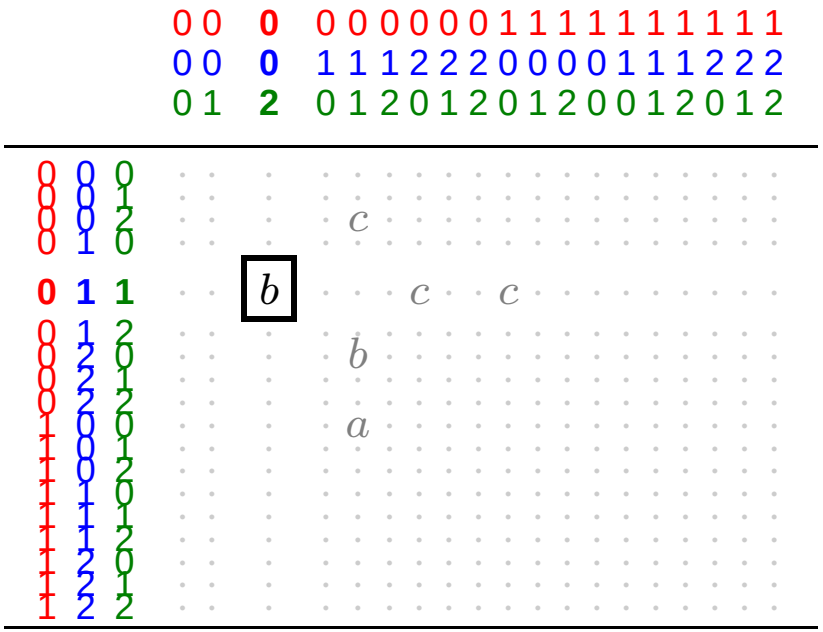
$Top(\alpha)$: *the highest MDD level affected by event* α

$$Top(a) \;=\; Top(d) \;=\; x \qquad\qquad Top(b) \;=\; Top(c) \;=\; y$$

B. Plateau, *On the stochastic structure of parallelism and synchronization models for distributed algorithm*, SIGMETRICS 1985

Kronecker encoding of the $\mathbf{N}$ $(K = 3)$

$$\mathbf{N} \;=\; \sum_{e\in\mathcal{E}} \mathbf{N}_e \;=\; \sum_{e\in\mathcal{E}} \bigotimes_{K\geq k\geq 1} \mathbf{N}_{k,e}$$



	a	b	c	d
x	$1 \to 0$	$\mathbf{I}$	$\mathbf{I}$	$0 \to 1$
y	$0 \to 1$	$\boxed{1} \to \boxed{0}$ $2 \to 1$	$0 \to 1$ $1 \to 2$	$1 \to 0$
z	$0 \to 1$	$0 \to 1$ $\boxed{1} \to \boxed{2}$	$2 \to 1$ $1 \to 0$	$1 \to 0$

$Top(\alpha)$: *the highest MDD level affected by event* α

$$Top(a) \;=\; Top(d) \;=\; x \qquad\qquad Top(b) \;=\; Top(c) \;=\; y$$

B. Plateau, *On the stochastic structure of parallelism and synchronization models for distributed algorithm*, SIGMETRICS 1985

Saturation-style fixed point iteration strategy

An MDD node at level k is ***saturated*** if the state set it encodes is a fixed point with respect to any event α such that $Top(\alpha) \leq k$ (α is independent of any MDD level higher than k)

- build the MDD encoding of initial state set
- From $k=1$ to K,

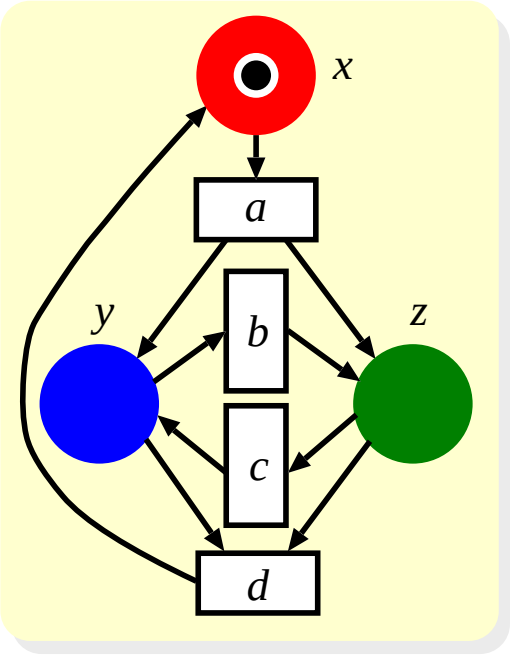
  saturate each node at level k by firing all events α s.t. $Top(\alpha)=k$

  (if this creates nodes below level k, saturate them immediately upon creation)
- When the root node is saturated, all reachable states have been discovered

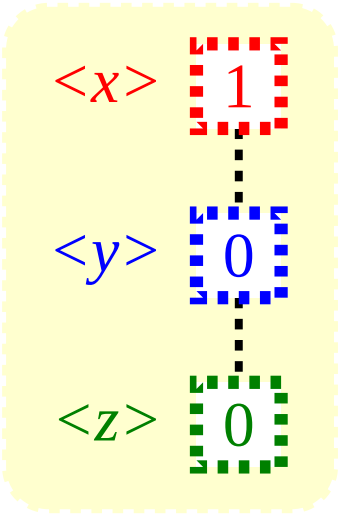
Saturation has Enormous time and memory efficiency compared to traditional breath-first iteration for asynchronous models

G. Ciardo, G. Lüettgen, and R. Siminiceanu, *Saturation : an efficient iteration strategy for symbolic state-space generation*, TACAS 2001

Setup initial configuration



	a	b	c	d
x	$1 \to 0$	**I**	**I**	$0 \to 1$
y	$0 \to 1$	$1 \to 0$ $2 \to 1$	$0 \to 1$ $1 \to 2$	$1 \to 0$
z	$0 \to 1$	$0 \to 1$ $1 \to 2$	$2 \to 1$ $1 \to 0$	$1 \to 0$

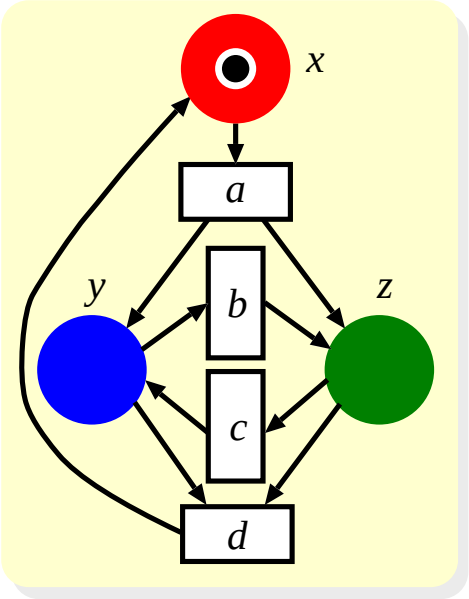


$$\mathcal{S}_p = \{p^0, p^1\} \equiv \{0, 1\}$$

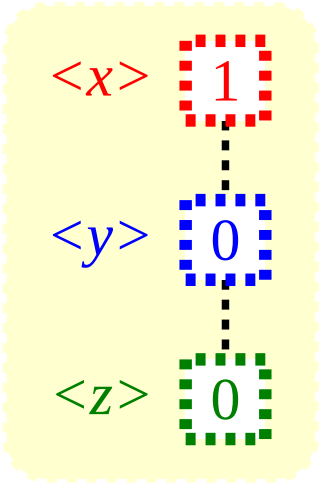
$$\mathcal{S}_q = \{q^0, q^1, q^2\} \equiv \{0, 1, 2\}$$

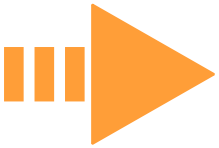
$$\mathcal{S}_r = \{r^0, r^1, r^2\} \equiv \{0, 1, 2\}$$

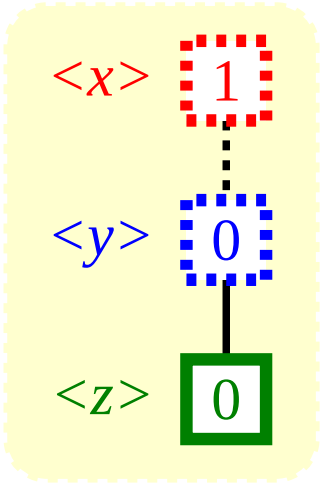
Saturate node $\boxed{0}$ at level $<z>$



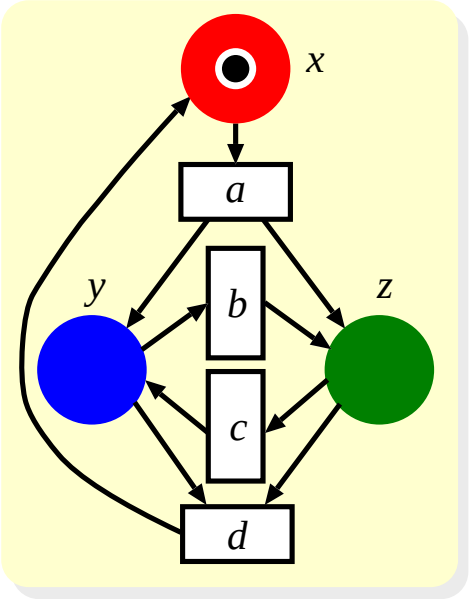
	a	b	c	d
x	$1 \to 0$	**I**	**I**	$0 \to 1$
y	$0 \to 1$	$1 \to 0$ $2 \to 1$	$0 \to 1$ $1 \to 2$	$1 \to 0$
z	$0 \to 1$	$0 \to 1$ $1 \to 2$	$2 \to 1$ $1 \to 0$	$1 \to 0$



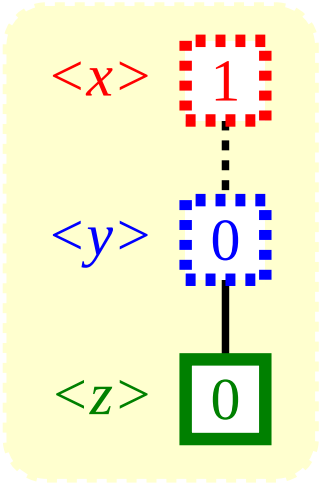


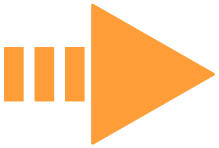


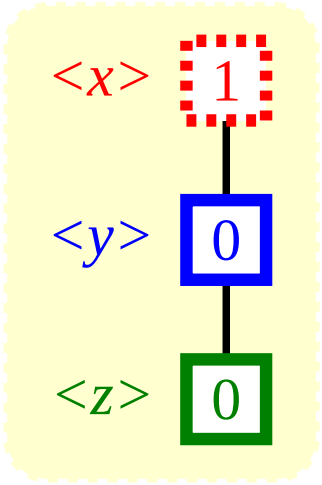
Saturate node $\boxed{0}$ at level $<y>$



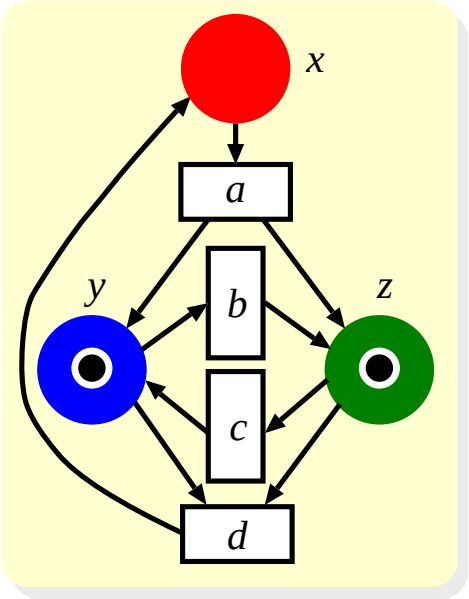
	a	$\boxed{b}$	$\boxed{c}$	d
x	$1 \to 0$	**I**	**I**	$0 \to 1$
y	$0 \to 1$	$1 \to 0$ $2 \to 1$	$\underline{0} \to \boxed{1}$ $1 \to 2$	$1 \to 0$
z	$0 \to 1$	$\underline{0} \to \boxed{1}$ $1 \to 2$	$2 \to 1$ $1 \to 0$	$1 \to 0$



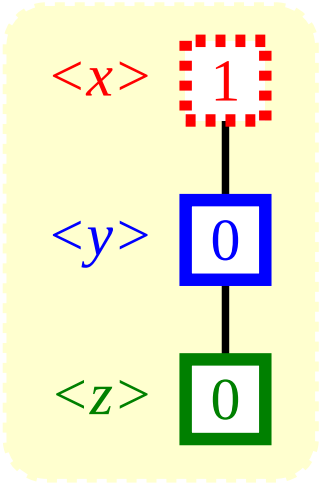


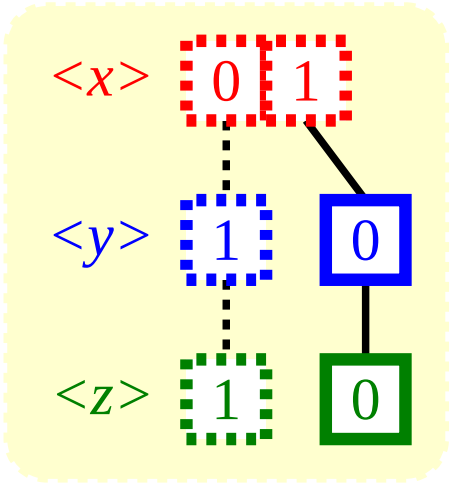


Saturate node $\boxed{1}$ at level $<x>$

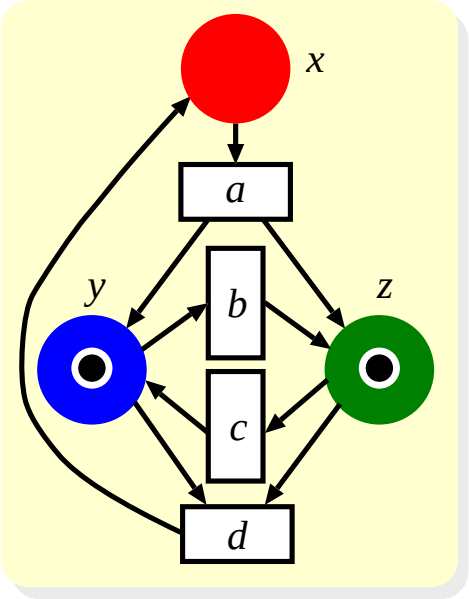


	$\boxed{a}$	b	c	d
x	$\underline{1} \to \boxed{0}$	**I**	**I**	$0 \to 1$
y	$\underline{0} \to \boxed{1}$	$1 \to 0$ $2 \to 1$	$0 \to 1$ $1 \to 2$	$1 \to 0$
z	$\underline{0} \to \boxed{1}$	$0 \to 1$ $1 \to 2$	$2 \to 1$ $1 \to 0$	$1 \to 0$

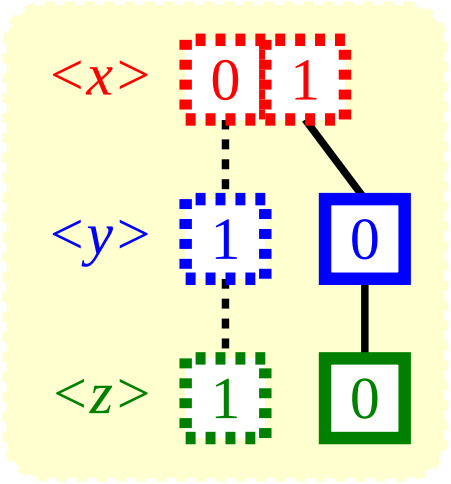


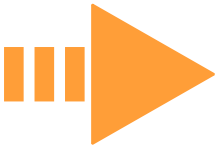


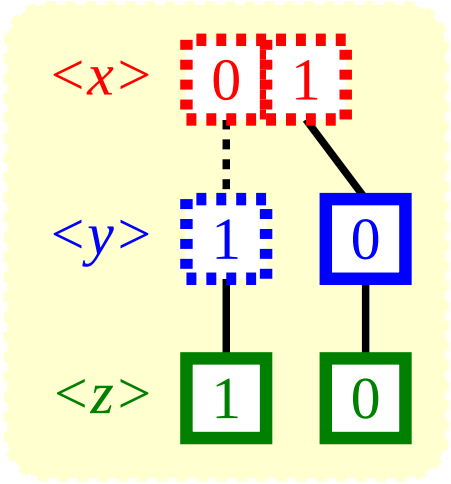
Saturate node $\boxed{1}$ at level $<z>$



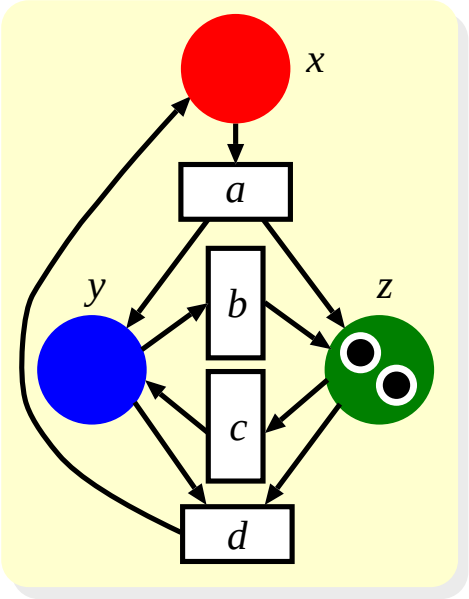
	a	b	c	d
x	$1 \to 0$	**I**	**I**	$0 \to 1$
y	$0 \to 1$	$1 \to 0$ $2 \to 1$	$0 \to 1$ $1 \to 2$	$1 \to 0$
z	$0 \to 1$	$0 \to 1$ $1 \to 2$	$2 \to 1$ $1 \to 0$	$1 \to 0$



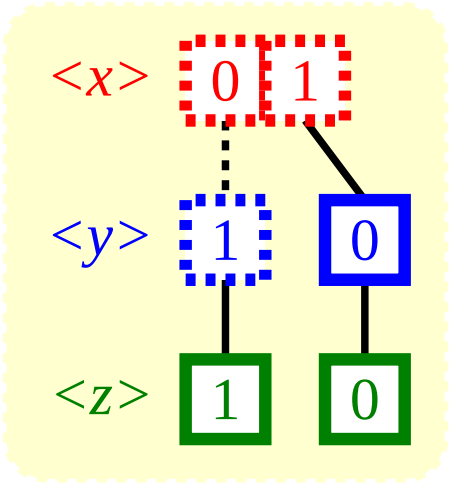


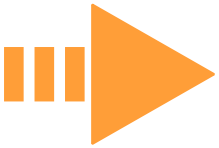


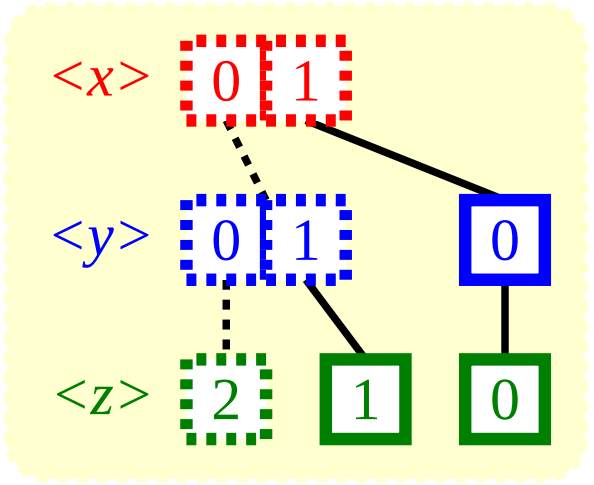
Saturate node $\boxed{1}$ at level $<y>$



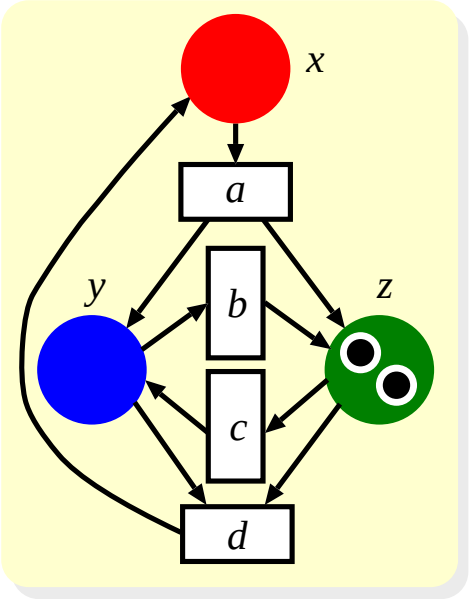
	a	$\boxed{b}$	c	d
x	$1 \to 0$	**I**	**I**	$0 \to 1$
y	$0 \to 1$	$\underline{1} \to \boxed{0}$ $2 \to 1$	$0 \to 1$ $1 \to 2$	$1 \to 0$
z	$0 \to 1$	$0 \to 1$ $\underline{1} \to \boxed{2}$	$2 \to 1$ $1 \to 0$	$1 \to 0$



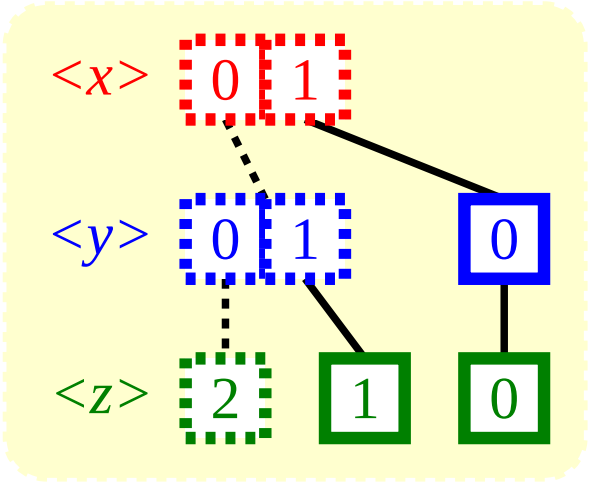


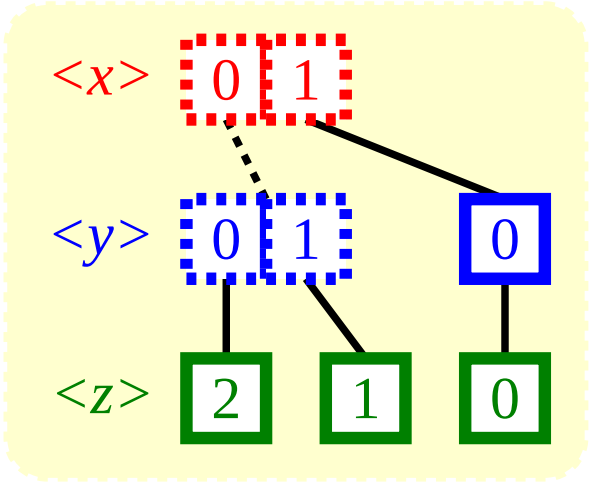


Saturate node $\boxed{2}$ at level $<z>$

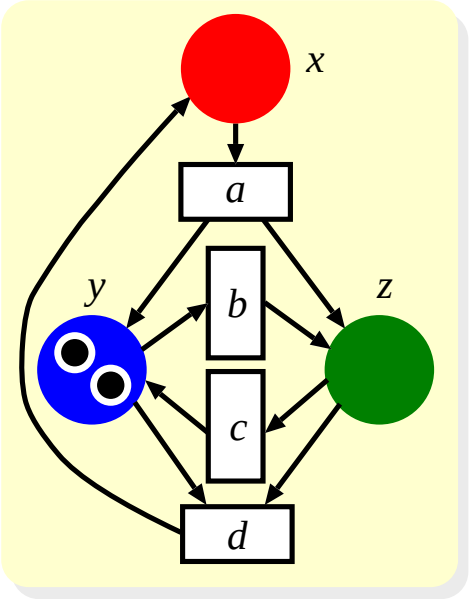


	a	b	c	d
x	$1 \to 0$	**I**	**I**	$0 \to 1$
y	$0 \to 1$	$1 \to 0$ $2 \to 1$	$0 \to 1$ $1 \to 2$	$1 \to 0$
z	$0 \to 1$	$0 \to 1$ $1 \to 2$	$2 \to 1$ $1 \to 0$	$1 \to 0$

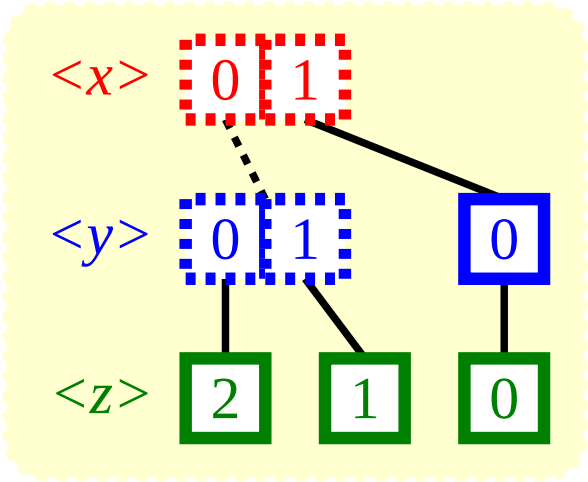


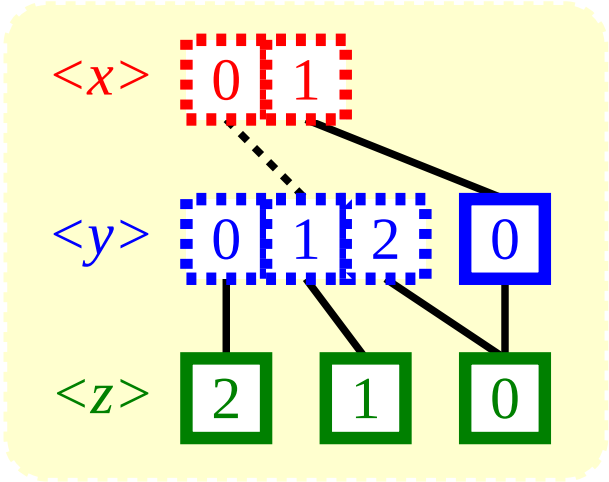


Saturate node $\boxed{0}\boxed{1}$ at level $<y>$

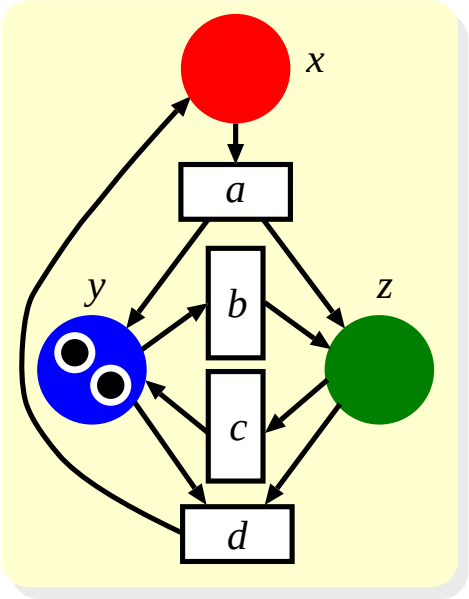


	a	b	$\boxed{c}$	d
x	$1 \to 0$	**I**	**I**	$0 \to 1$
y	$0 \to 1$	$1 \to 0$ $2 \to 1$	$0 \to 1$ $\underline{1} \to \boxed{2}$	$1 \to 0$
z	$0 \to 1$	$0 \to 1$ $1 \to 2$	$2 \to 1$ $\underline{1} \to \boxed{0}$	$1 \to 0$

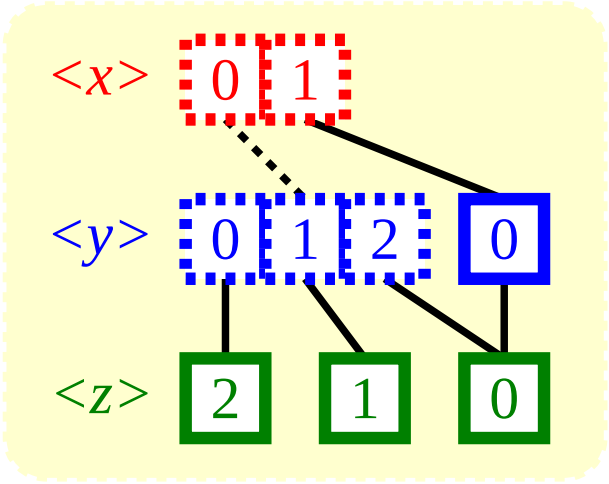


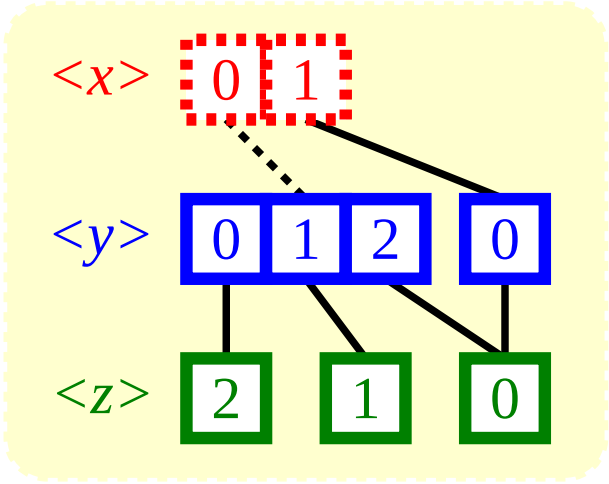


Saturate node 0 1 2 at level $<y>$

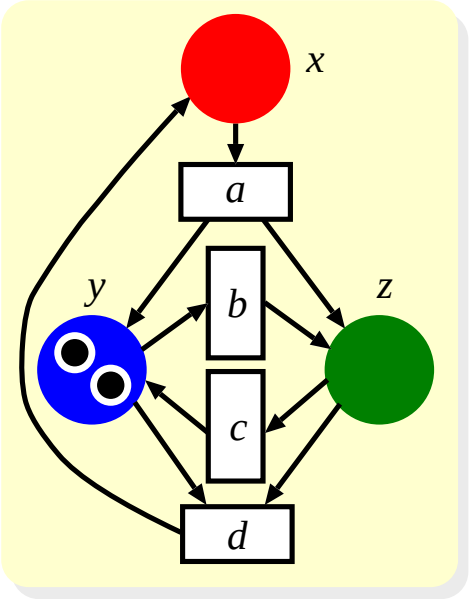


	a	b	c	d
x	$1 \to 0$	**I**	**I**	$0 \to 1$
y	$0 \to 1$	$1 \to 0$ $2 \to 1$	$0 \to 1$ $1 \to 2$	$1 \to 0$
z	$0 \to 1$	$0 \to 1$ $1 \to 2$	$2 \to 1$ $1 \to 0$	$1 \to 0$

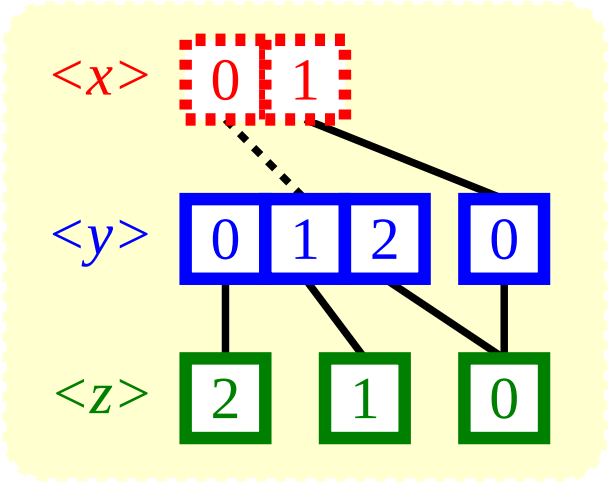


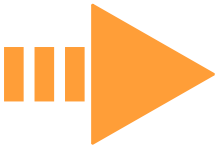


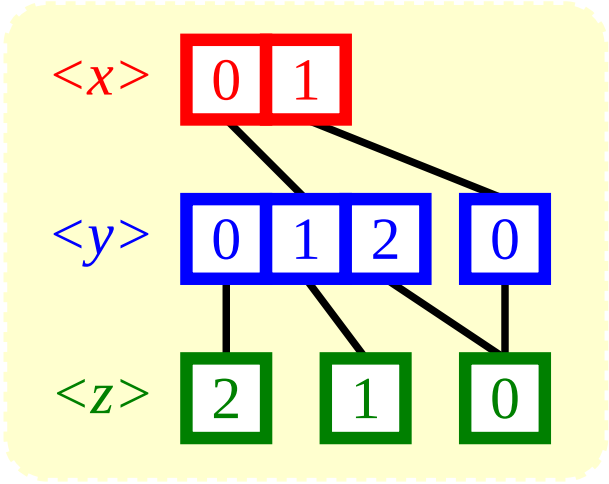
Saturate node $\boxed{0}\boxed{1}$ at level $<x>$



	a	b	c	d
x	$1 \to 0$	**I**	**I**	$0 \to 1$
y	$0 \to 1$	$1 \to 0$ $2 \to 1$	$0 \to 1$ $1 \to 2$	$1 \to 0$
z	$0 \to 1$	$0 \to 1$ $1 \to 2$	$2 \to 1$ $1 \to 0$	$1 \to 0$







Saturation NOW

- **Problem** :

  We target relatively large models where state-space generation with a single workstation must rely on virtual memory or runs out of memory

- **Solution** :
  - Apply the saturation algorithm on a network of workstations to increase the available memory
  - *Horizontal MDDs slicing for distributed state-space construction* :
    - no duplicate MDDs
    - no global synchronization
  - *Nested approach of dynamic memory load balancing scheme* :
    - pairwise communication to achieve nearly ideal memory scalability

Pattern Recognition Approach of Speculative Firing Prediction

Speculative firing prediction

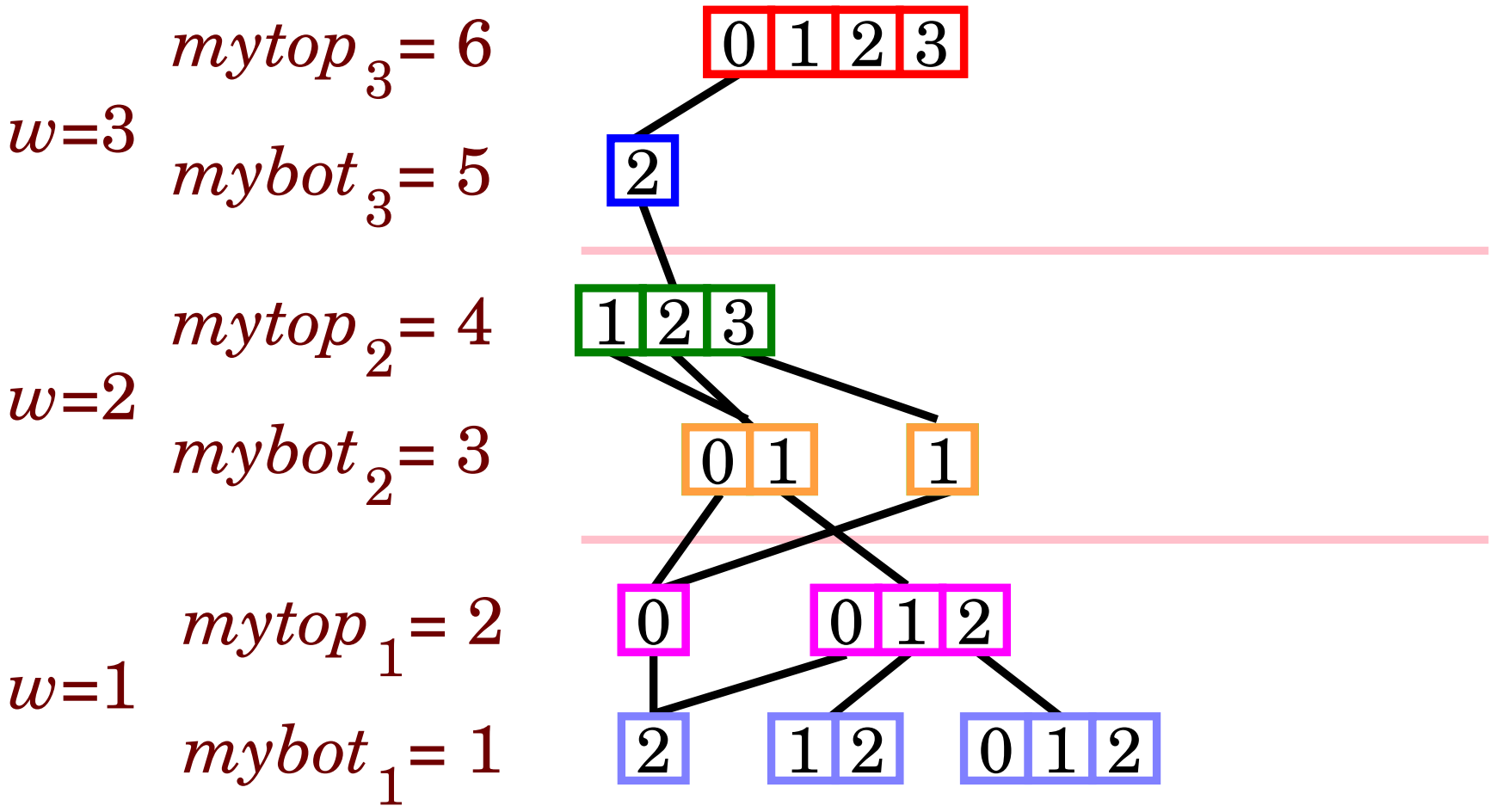
- **Problem** :
  - *Only one workstation is active* at any time :

    no theoretical speedup is achieved
  - The horizontal slicing scheme *sequentializes the distributed computation* :

    parallelization becomes a challenge now

- **Solution** :
  - *Idle workstations perform some event firings in advance* and cache the results of those firings
  - Some later firing requests might retrieve those results from caches and be resolved quickly

Speculative event firing



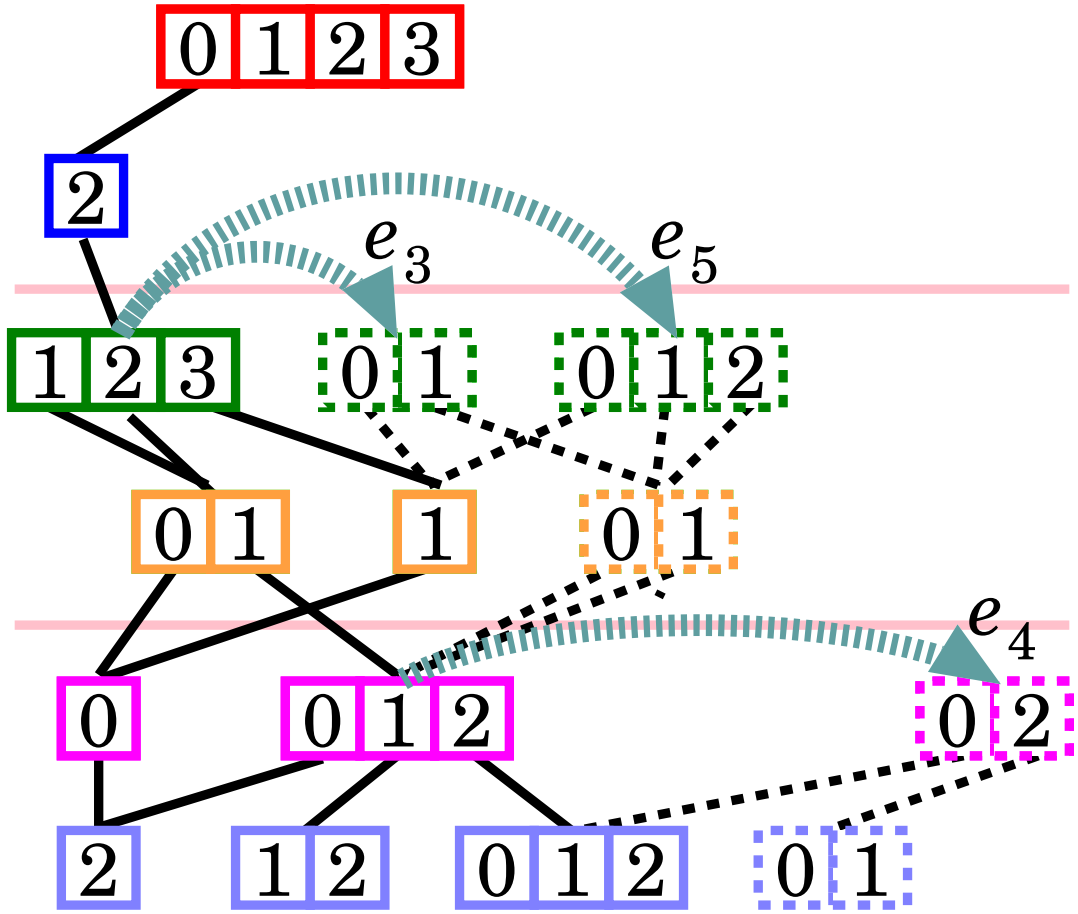
A runtime snapshot

Local event firing prediction

w=3 $mytop_3$= 6 $mybot_3$= 5

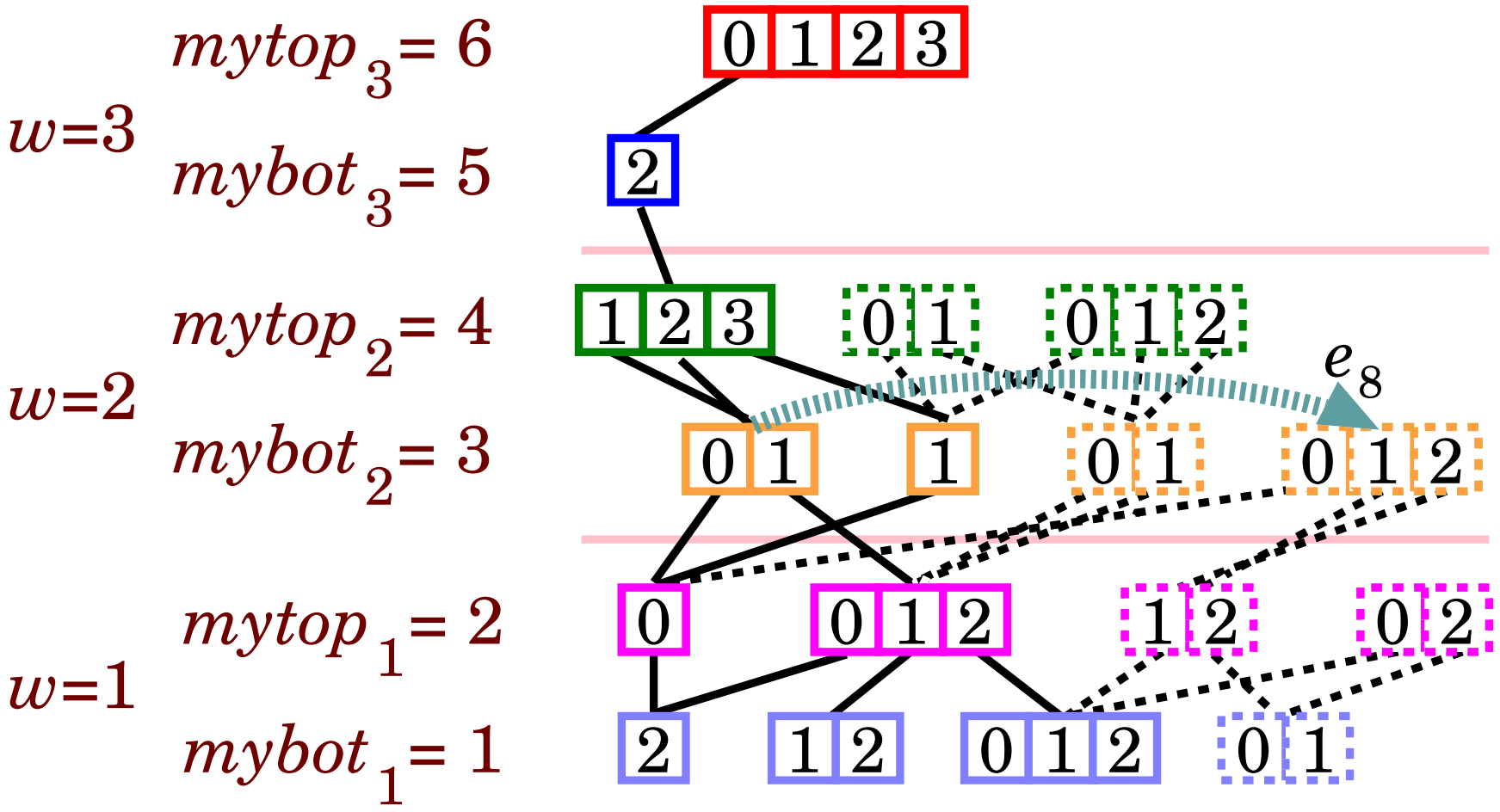
w=2 $mytop_2$= 4 $mybot_2$= 3

w=1 $mytop_1$= 2 $mybot_1$= 1



w_2 and w_1 perform speculative event fi rings individually

Non-local event firing prediction



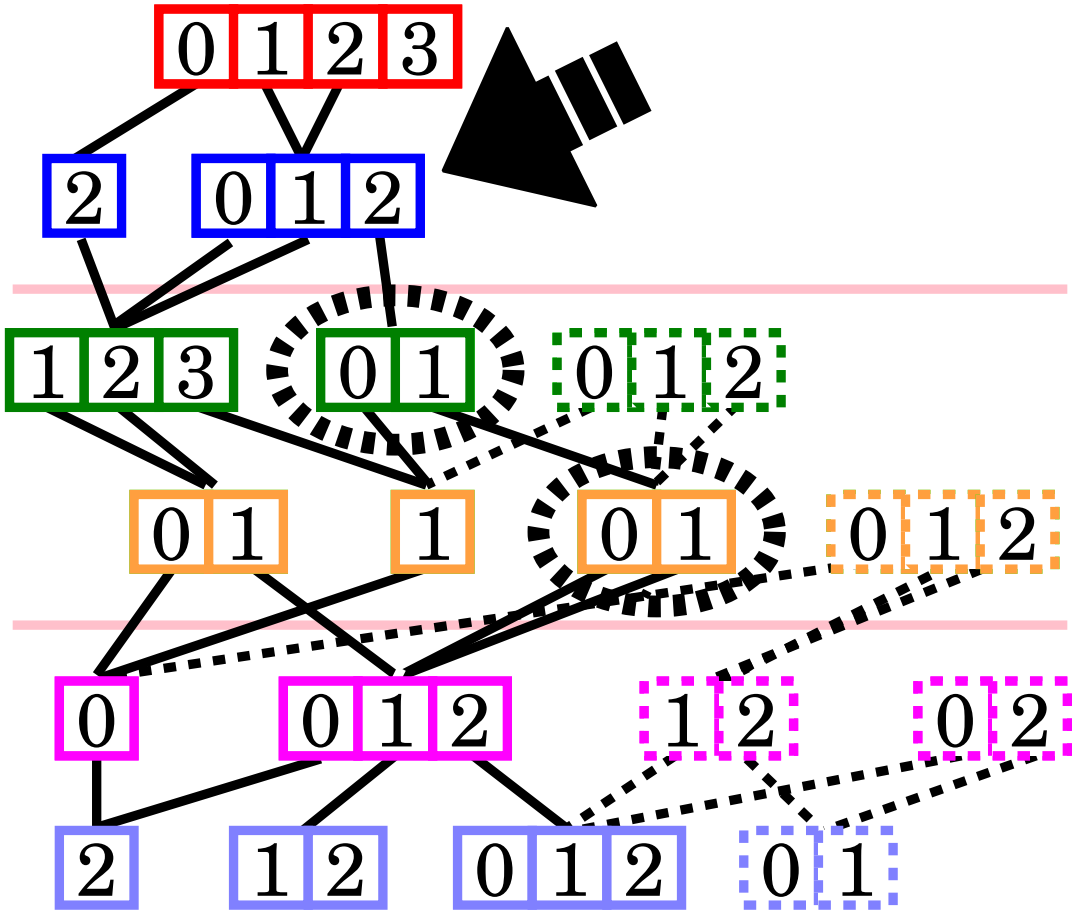
w_2 and w_1 perform some speculative event fi ring together

Local event firing

w=3 $mytop_3$= 6, $mybot_3$= 5

w=2 $mytop_2$= 4, $mybot_2$= 3

w=1 $mytop_1$= 2, $mybot_1$= 1



w_3 retrieves event fi ring result from operation cache

Remote event firing request

w=3

$mytop_3$= 6

$mybot_3$= 5

w=2

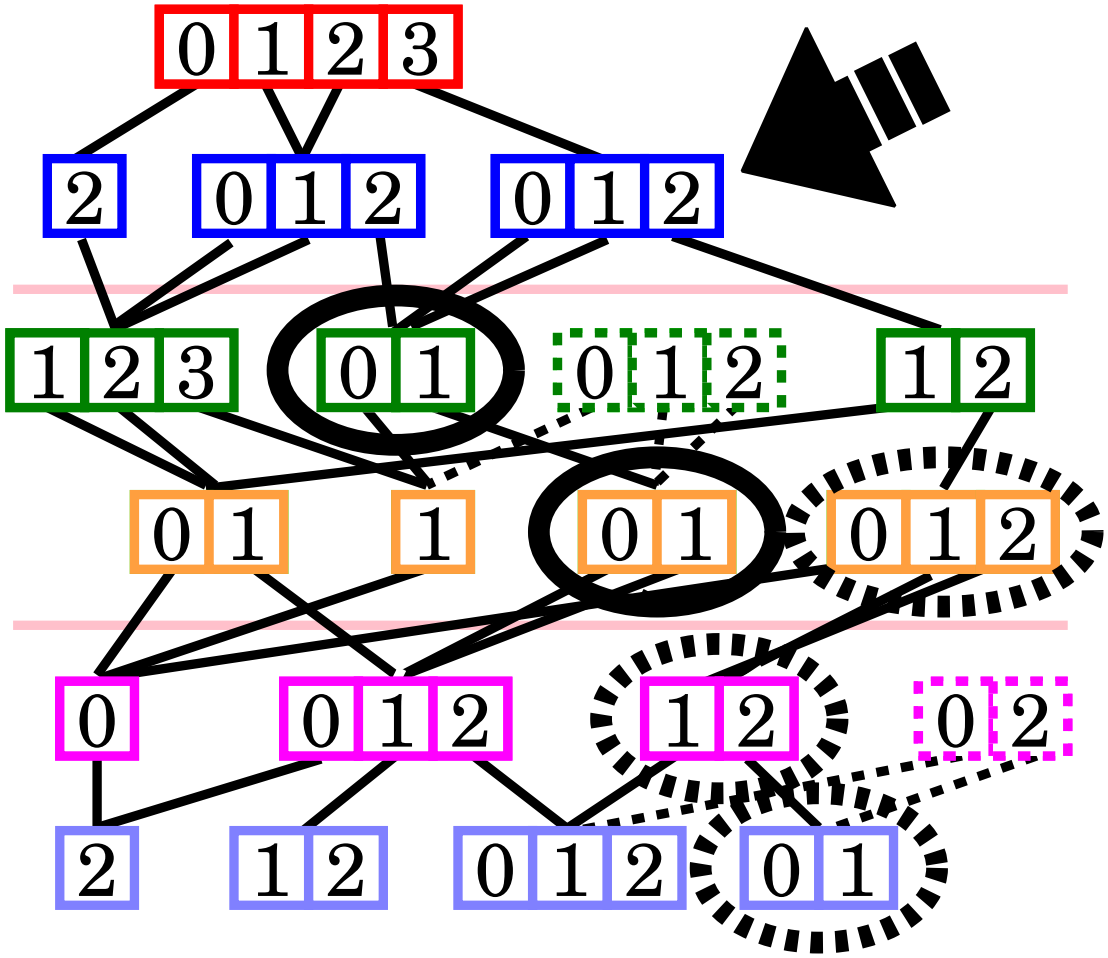
$mytop_2$= 4

$mybot_2$= 3

w=1

$mytop_1$= 2

$mybot_1$= 1



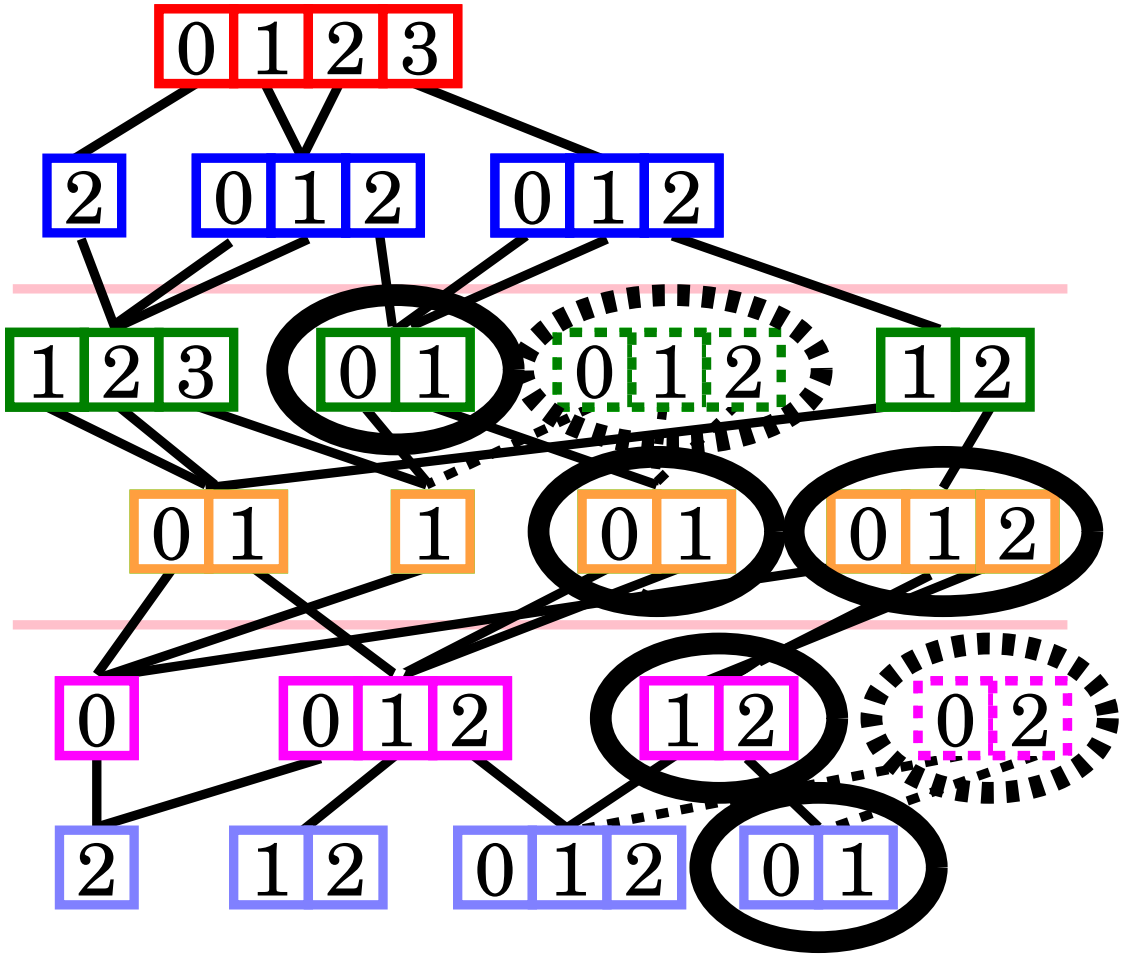
w_3 makes event fi ring requests remotely

Drawbacks

w=3 $mytop_3 = 6$, $mybot_3 = 5$

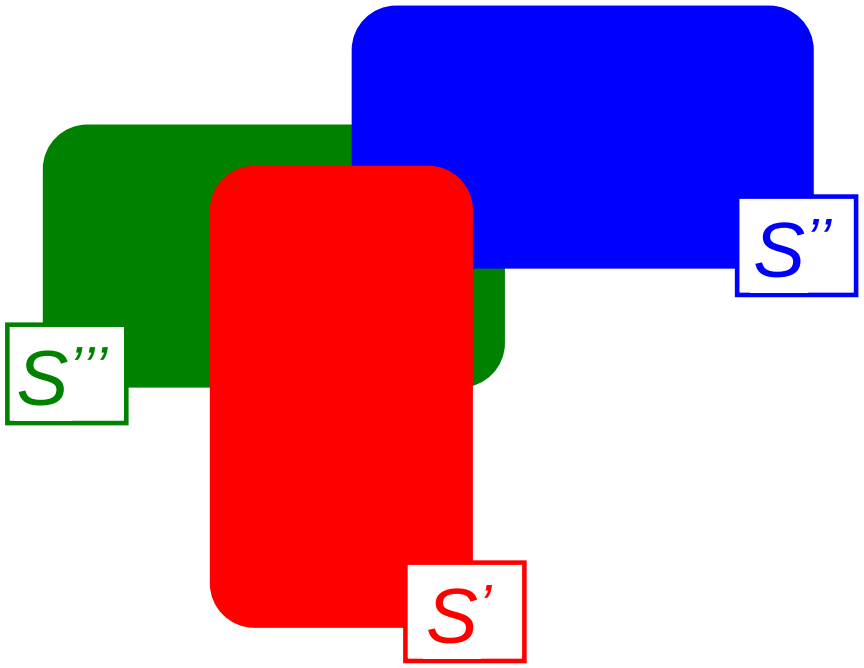
w=2 $mytop_2 = 4$, $mybot_2 = 3$

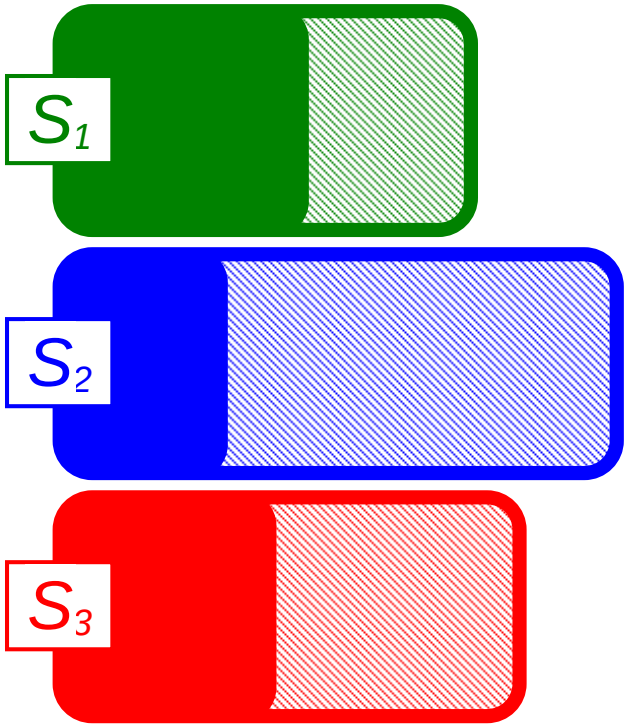
w=1 $mytop_1 = 2$, $mybot_1 = 1$



- Space for storing the disconnected MDDs
- Space for storing the additional caches entries

Vertical slicing scheme vs. firing prediction





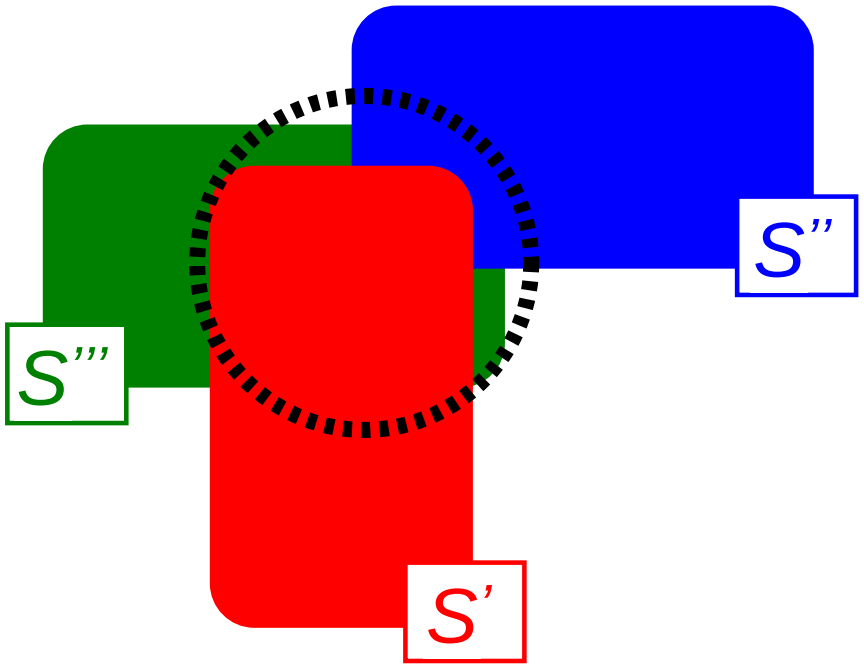
Overlapped BDDs

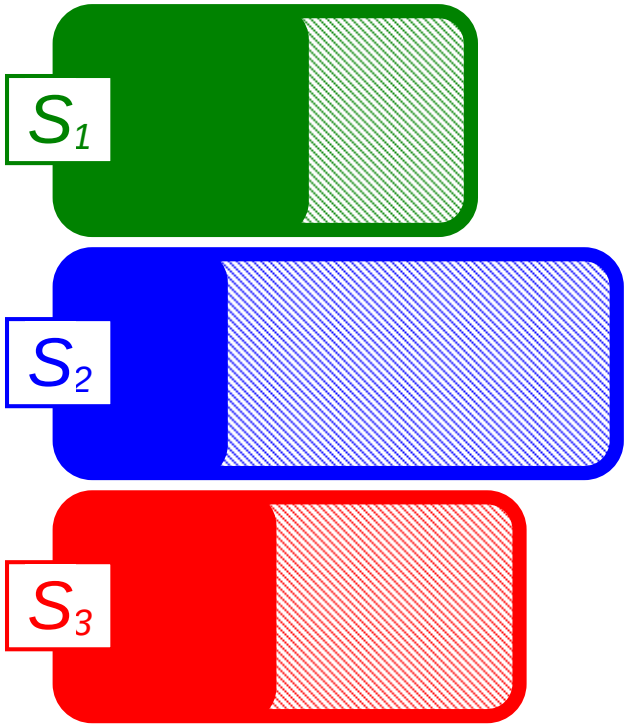
(Duplicate)

Disconnected MDDs

(Useless)

Vertical slicing scheme vs. firing prediction





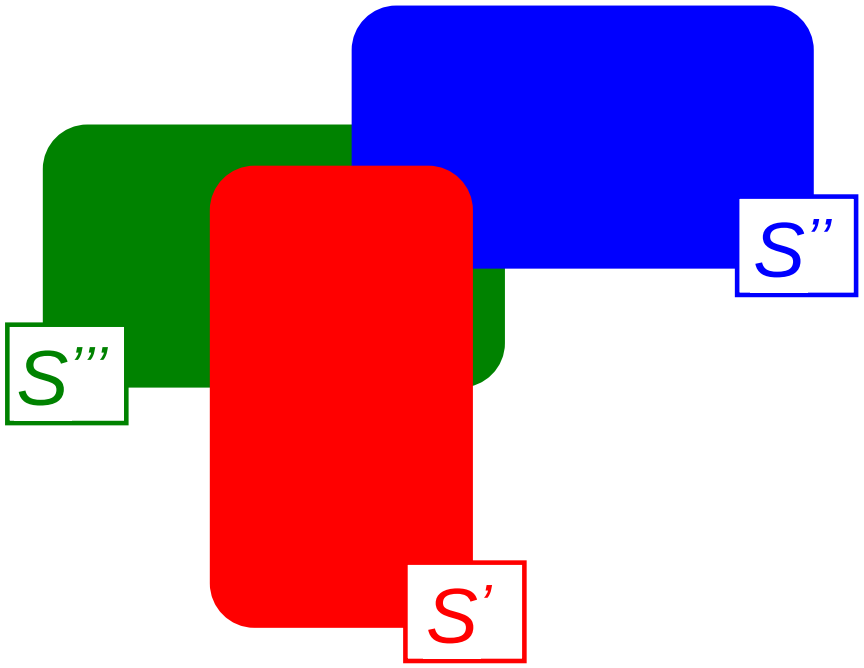
Overlapped BDDs

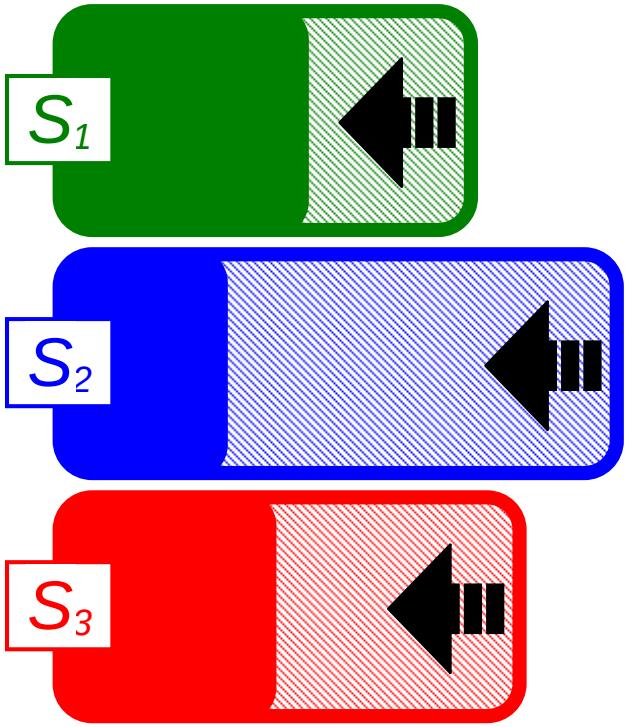
(Duplicate)

Disconnected MDDs

(Useless)

Vertical slicing scheme vs. firing prediction





Overlapped BDDs

(Duplicate)

Disconnected MDDs

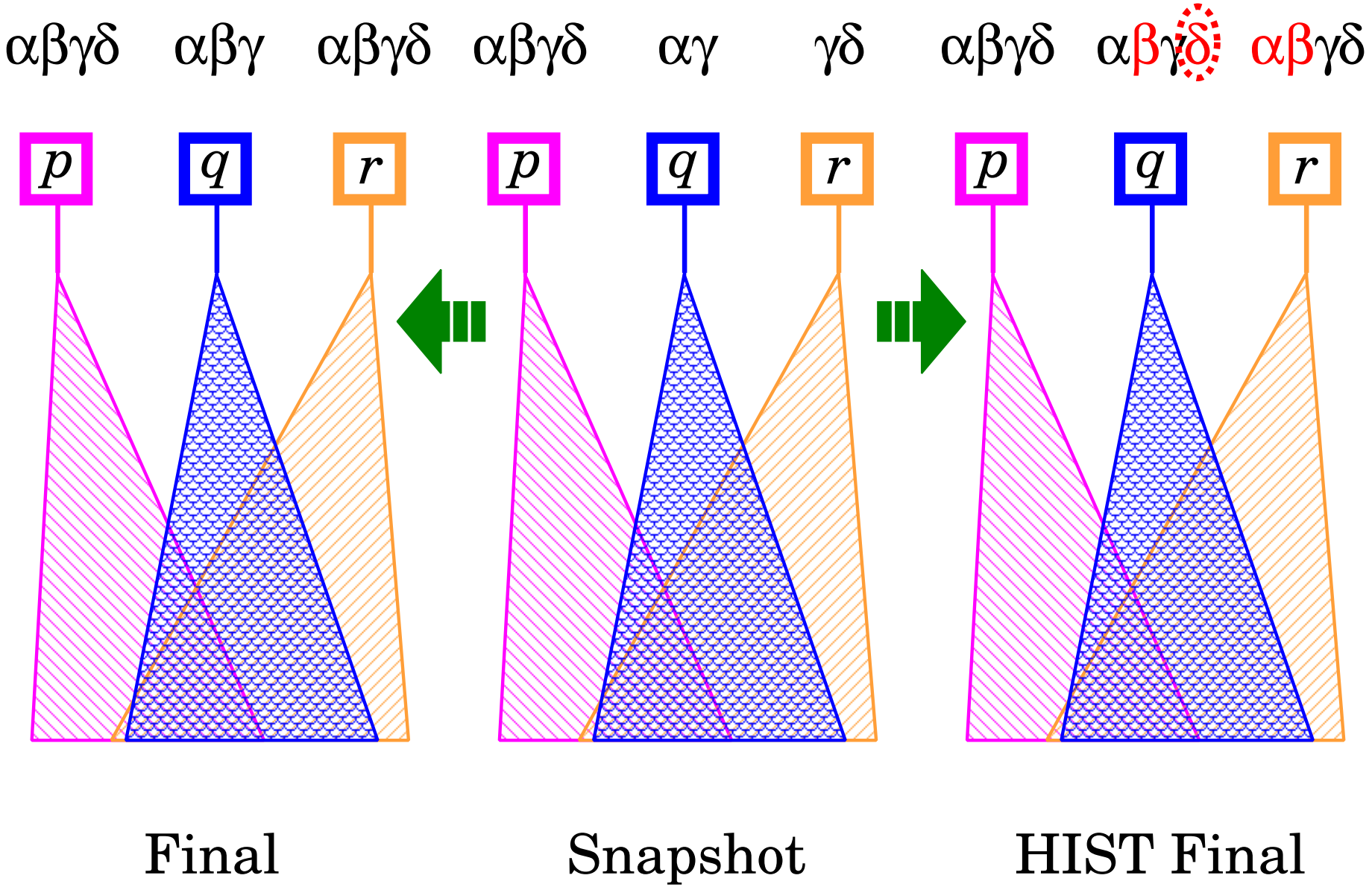
(Useless)

Pattern recognition approach

- **Problem** :
  - Do not know a priori whether an event will be fired on some MDD node
  - A NAÏVE approach : *idle workstations exhaust all possible firings*

    $\mathcal{E}_{all}(p) = \{e : Top(p.lvl) > k \geq Bot(e)\}$ for each node p
    - It might create many useless nodes and firing cache entries
    - It will not stop when the prediction does not help much
- **Observation** :
  - $\mathcal{E}_{patt}(p) = \{e$: event e need to be fired on the node p to saturate $p\}$
  - There are always some MDD nodes which have similar or even the same set of events fired on.
- **Solution** :
  - A more informed firing prediction based on *firing patterns*

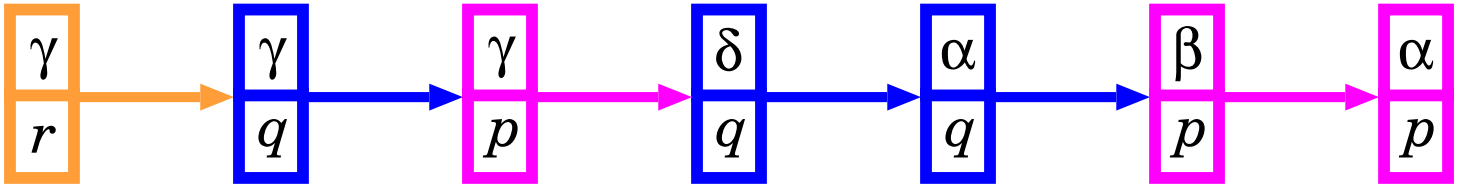
Pattern recognition approach



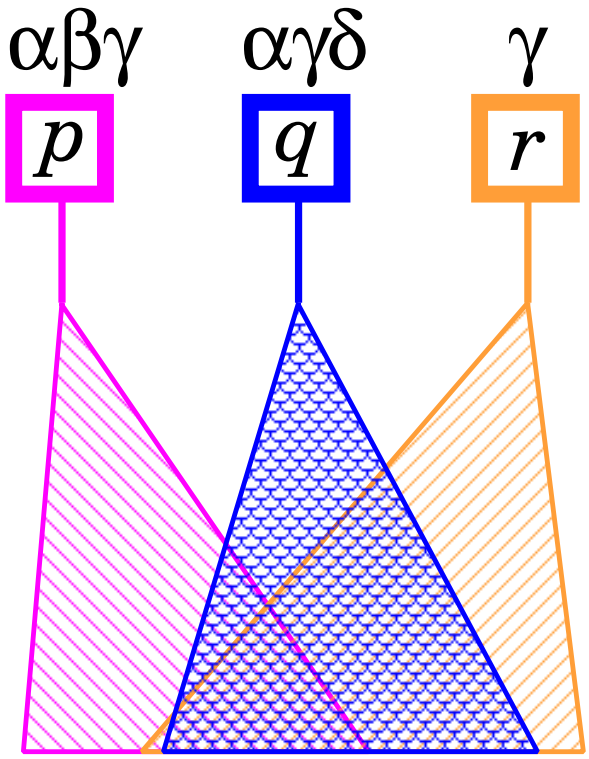
A runtime snapshot of $\mathcal{E}_{hist}(p)$, $\mathcal{E}_{hist}(q)$, and $\mathcal{E}_{hist}(r)$

$Requests$

(stack)




```
FirePredict(Requests : stack, Class : array)
 while Requests ≠ ∅ do
   (e, p) ← Pop(Requests);
   Enqueue(e, E_hist(p));
   q ← Class_{p.lvl}[e];
   if E_hist(p) ⊃ E_hist(q) then
     Class_{p.lvl}[e] ← p;
     fire e on q and cache the result;
   else if E_hist(p) ⊂ E_hist(q) then
     foreach e' ∈ E_hist(q) \ E_hist(p) do
       fire e' on p and cache the result;
```

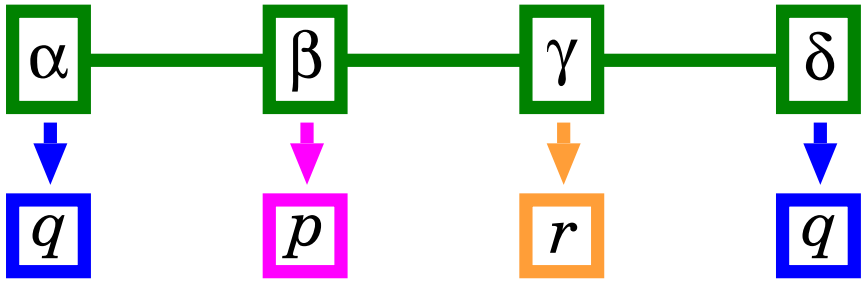


Pop one element of $Requests$ in each iteration

A runtime snapshot of $Class$

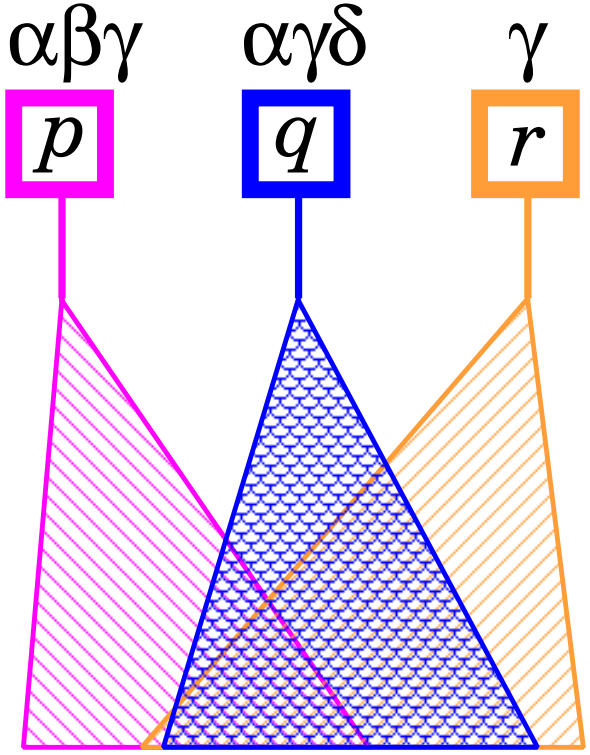
$Class_{p.lvl}$

(array)



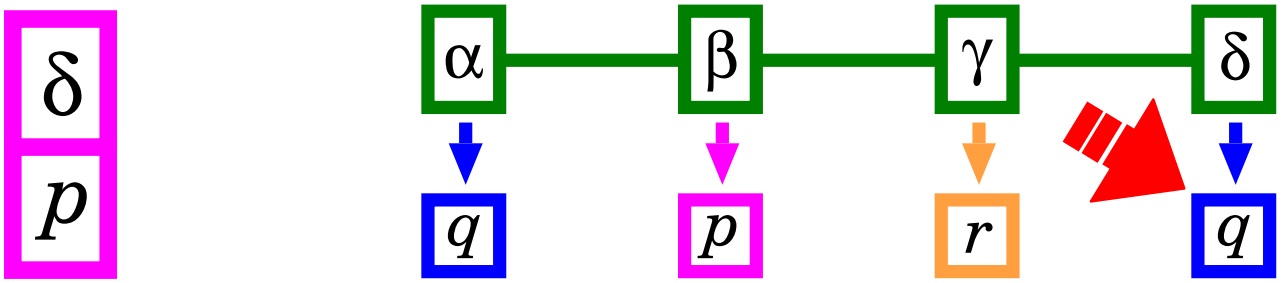
$FirePredict(Requests : stack, Class : array)$

while $Requests \neq \emptyset$ **do**
  $(e, p) \leftarrow Pop(Requests)$;
  $Enqueue(e, \mathcal{E}_{hist}(p))$;
  $q \leftarrow Class_{p.lvl}[e]$;
  if $\mathcal{E}_{hist}(p) \supset \mathcal{E}_{hist}(q)$ **then**
    $Class_{p.lvl}[e] \leftarrow p$;
    fire e on q and cache the result;
  else if $\mathcal{E}_{hist}(p) \subset \mathcal{E}_{hist}(q)$ **then**
    foreach $e' \in \mathcal{E}_{hist}(q) \setminus \mathcal{E}_{hist}(p)$ **do**
      fire e' on p and cache the result;



For each k, $Class_k[e]$ indicate the representative node w.r.t. e

For (δ, p) entry



$FirePredict(Requests : stack, Class : array)$

while $Requests \neq \emptyset$ **do**

$(e, p) \leftarrow Pop(Requests);$

$Enqueue(e, \mathcal{E}_{hist}(p));$

$q \leftarrow Class_{p.lvl}[e];$

if $\mathcal{E}_{hist}(p) \supset \mathcal{E}_{hist}(q)$ **then**

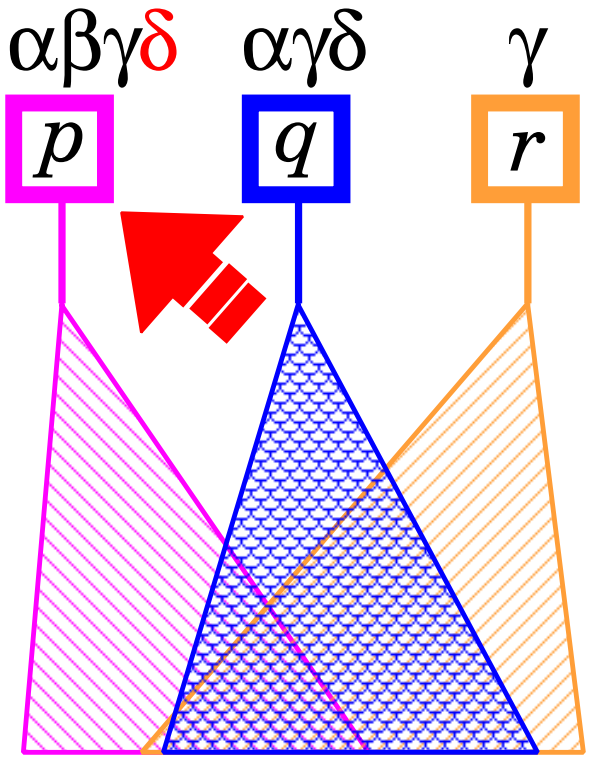
$Class_{p.lvl}[e] \leftarrow p;$

fire e on q and cache the result;

else if $\mathcal{E}_{hist}(p) \subset \mathcal{E}_{hist}(q)$ **then**

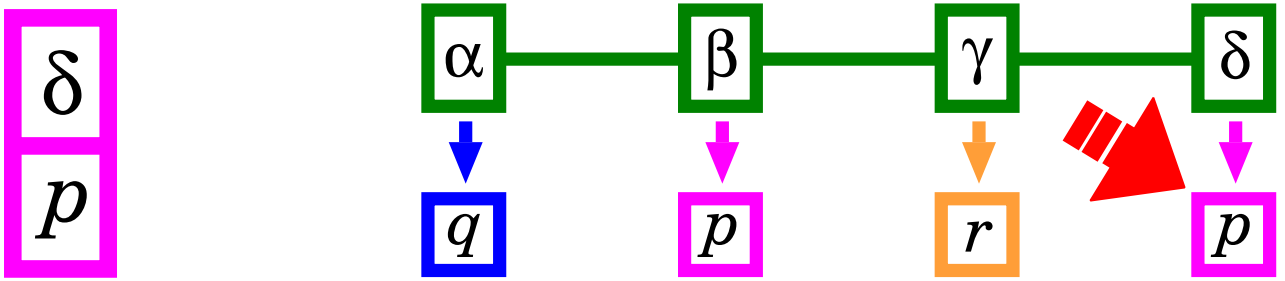
foreach $e' \in \mathcal{E}_{hist}(q) \setminus \mathcal{E}_{hist}(p)$ **do**

fire e' on p and cache the result;



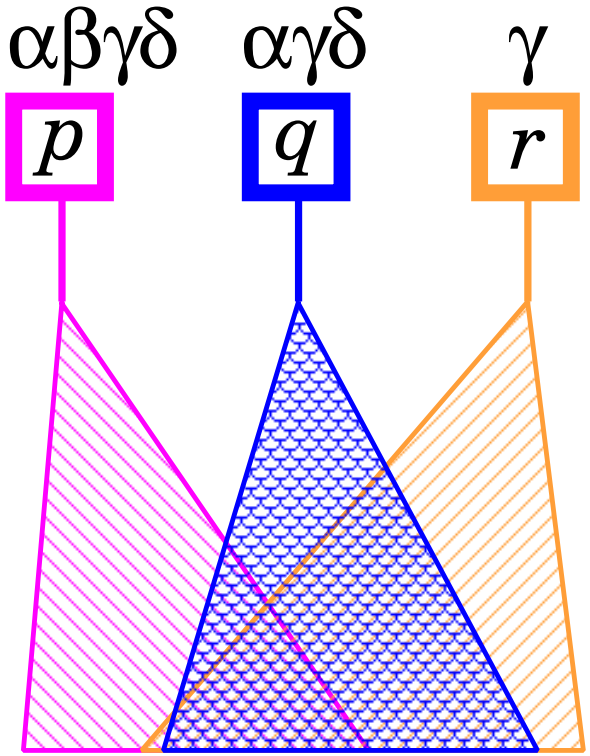
Pop (δ, p) and enqueue δ into $\mathcal{E}_{hist}(p)$

Updating the representative node w.r.t. δ



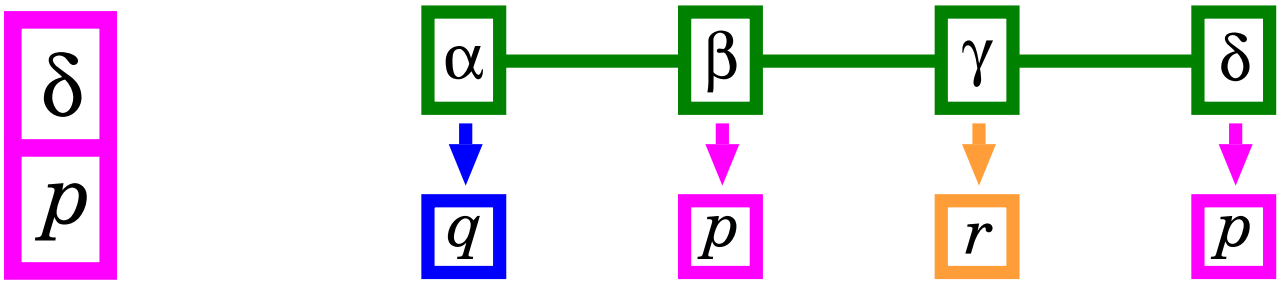
$FirePredict(Requests : stack, Class : array)$

while $Requests \neq \emptyset$ **do**
  $(e, p) \leftarrow Pop(Requests)$;
  $Enqueue(e, \mathcal{E}_{hist}(p))$;
  $q \leftarrow Class_{p.lvl}[e]$;
  if $\mathcal{E}_{hist}(p) \supset \mathcal{E}_{hist}(q)$ **then**
    $Class_{p.lvl}[e] \leftarrow p$;
    fire e on q and cache the result;
  else if $\mathcal{E}_{hist}(p) \subset \mathcal{E}_{hist}(q)$ **then**
    foreach $e' \in \mathcal{E}_{hist}(q) \setminus \mathcal{E}_{hist}(p)$ **do**
      fire e' on p and cache the result;



Set p as the new representative node of δ

Performing firing prediction



$FirePredict(Requests : stack, Class : array)$

while $Requests \neq \emptyset$ **do**

$(e, p) \leftarrow Pop(Requests)$;

$Enqueue(e, \mathcal{E}_{hist}(p))$;

$q \leftarrow Class_{p.lvl}[e]$;

if $\mathcal{E}_{hist}(p) \supset \mathcal{E}_{hist}(q)$ **then**

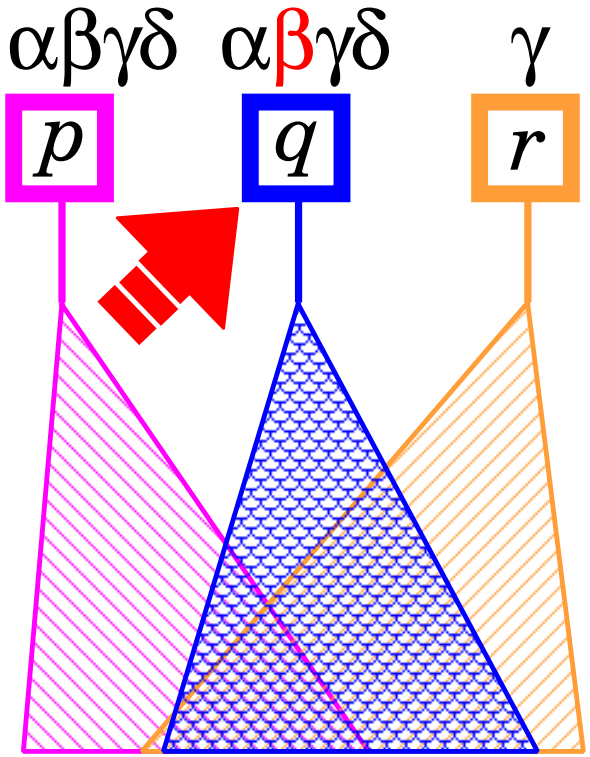
$Class_{p.lvl}[e] \leftarrow p$;

fire e on q and cache the result;

else if $\mathcal{E}_{hist}(p) \subset \mathcal{E}_{hist}(q)$ **then**

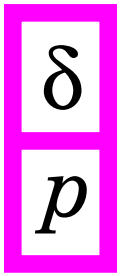
foreach $e' \in \mathcal{E}_{hist}(q) \setminus \mathcal{E}_{hist}(p)$ **do**

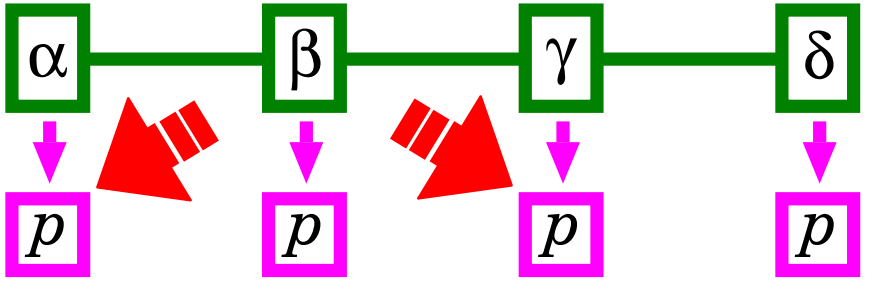
fire e' on p and cache the result;



Speculatively fire β on q

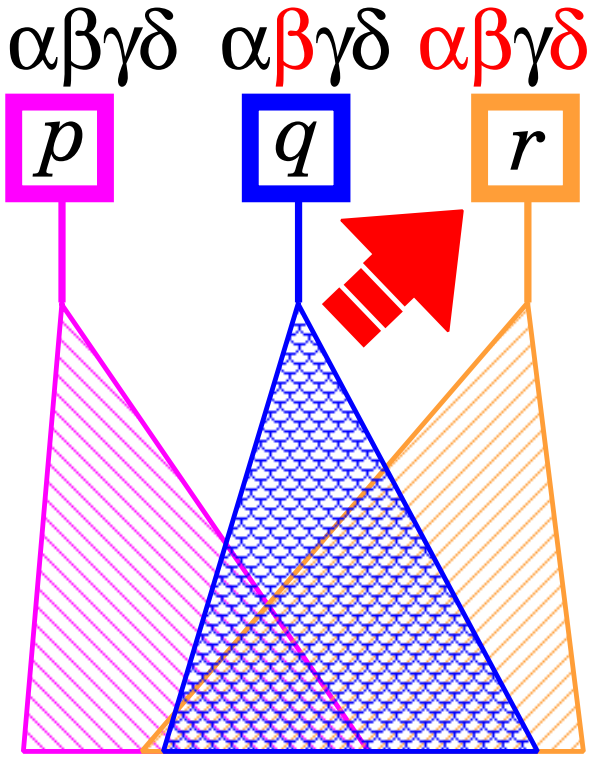
An alternative approach





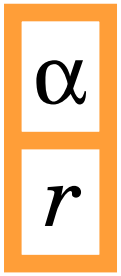
$FirePredict(Requests : stack, Class : array)$

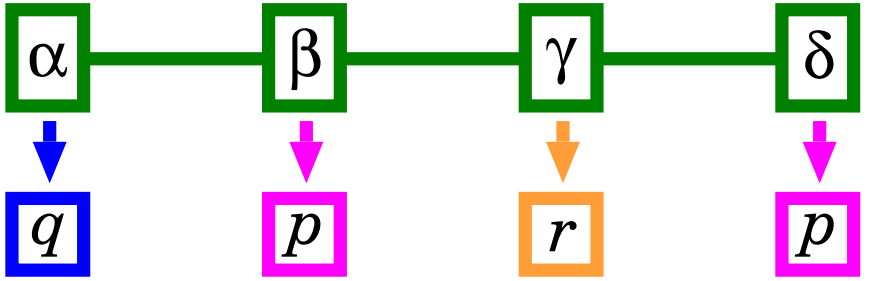
while $Requests \neq \emptyset$ **do**
  $(e, p) \leftarrow Pop(Requests)$;
  $Enqueue(e, \mathcal{E}_{hist}(p))$;
  $q \leftarrow Class_{p.lvl}[e]$;
  if $\mathcal{E}_{hist}(p) \supset \mathcal{E}_{hist}(q)$ **then**
    $Class_{p.lvl}[e] \leftarrow p$;
    fire e on q and cache the result;
  else if $\mathcal{E}_{hist}(p) \subset \mathcal{E}_{hist}(q)$ **then**
    foreach $e' \in \mathcal{E}_{hist}(q) \setminus \mathcal{E}_{hist}(p)$ **do**
      fire e' on p and cache the result;



Aggressively update and predict the firings of α, β, and δ on r

Firing prediction scenario 1





$FirePredict(Requests : stack, Class : array)$

while $Requests \neq \emptyset$ **do**

$(e, p) \leftarrow Pop(Requests)$;

$Enqueue(e, \mathcal{E}_{hist}(p))$;

$q \leftarrow Class_{p.lvl}[e]$;

if $\mathcal{E}_{hist}(p) \supset \mathcal{E}_{hist}(q)$ **then**

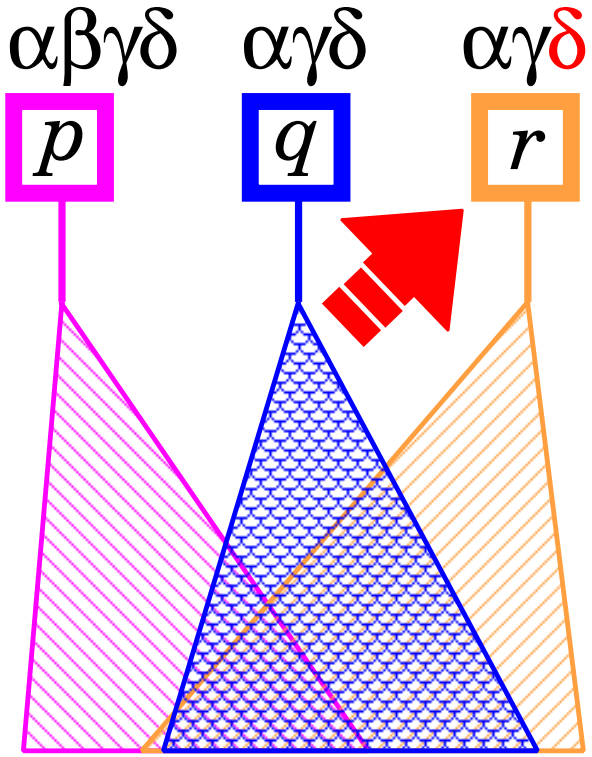
$Class_{p.lvl}[e] \leftarrow p$;

fire e on q and cache the result;

else if $\mathcal{E}_{hist}(p) \subset \mathcal{E}_{hist}(q)$ **then**

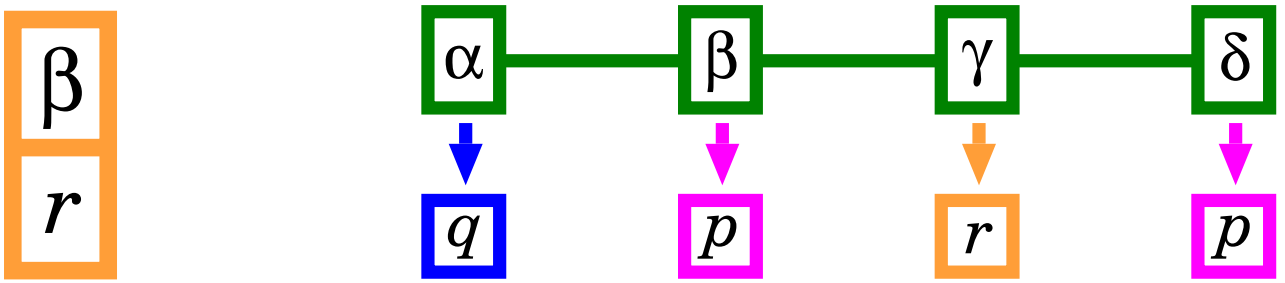
foreach $e' \in \mathcal{E}_{hist}(q) \setminus \mathcal{E}_{hist}(p)$ **do**

fire e' on p and cache the result;



Pop (α, r), enqueue α, and speculatively fire δ on r

Firing prediction scenario 2



FirePredict(*Requests* : *stack*, *Class* : *array*)

while $Requests \neq \emptyset$ **do**

$(e, p) \leftarrow Pop(Requests)$;

$Enqueue(e, \mathcal{E}_{hist}(p))$;

$q \leftarrow Class_{p.lvl}[e]$;

if $\mathcal{E}_{hist}(p) \supset \mathcal{E}_{hist}(q)$ **then**

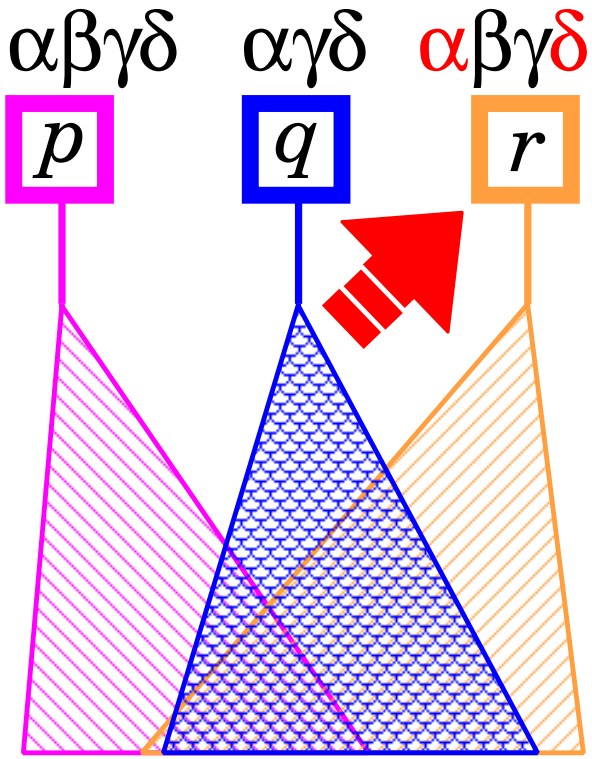
$Class_{p.lvl}[e] \leftarrow p$;

fire e on q and cache the result;

else if $\mathcal{E}_{hist}(p) \subset \mathcal{E}_{hist}(q)$ **then**

foreach $e' \in \mathcal{E}_{hist}(q) \setminus \mathcal{E}_{hist}(p)$ **do**

fire e' on p and cache the result;



Pop (β, r), enqueue β, and speculatively fire α and δ on r

Experimental Results

Slotted ring network protocol

- Protocol for local area networks : N is the number of nodes within the network ($K = N$, $|\mathcal{S}_k| = 10$ for all k, $|\mathcal{E}| = 3N$)

  E. Pastor, O. Roig, J. Cortadella, and R. Badia, *Petri net analysis using boolean manipulation*, ICATPN 1994

W	**Time** (sec)			**Total Memory** (MB)			**Max Memory** (MB)		
	SMART$^{\text{NOW}}$	NAÏVE	HIST	SMART$^{\text{NOW}}$	NAÏVE	HIST	SMART$^{\text{NOW}}$	NAÏVE	HIST
$N = 200$ $\quad \lvert\mathcal{S}\rvert = 8.38 \cdot 10^{211}$						SMART completes in 108 sec using 284MB			
2	119	-24%	-13%	286	$+3\%$	$+45\%$	197	$+1\%$	$+53\%$
4	139	-27%	-15%	286	$+11\%$	$+51\%$	127	$+61\%$	$+58\%$
8	182	-32%	-24%	286	$+129\%$	$+62\%$	69	$+239\%$	$+62\%$
$N = 300$ $\quad \lvert\mathcal{S}\rvert = 8.38 \cdot 10^{211}$						SMART does not complete in 5 hrs using 512MB			
2	$^{s}552$	$^{s}+5\%$	$^{s}-5\%$	962	$+25\%$	$+11\%$	562	$+8\%$	$+7\%$
4	$^{d}490$	> 5hrs	$^{d}-16\%$	962	-	$+34\%$	352	-	$+12\%$
8	564	> 5hrs	-39%	962	-	$+50\%$	252	-	$+23\%$

- Both NAÏVE and HIST approaches outperform SMART$^{\text{NOW}}$

Slotted ring network protocol

- Protocol for local area networks : N is the number of nodes within the network ($K = N$, $|\mathcal{S}_k| = 10$ for all k, $|\mathcal{E}| = 3N$)

  E. Pastor, O. Roig, J. Cortadella, and R. Badia, *Petri net analysis using boolean manipulation*, ICATPN 1994

W	**Time** (sec)			**Total Memory** (MB)			**Max Memory** (MB)		
	SMART$^{\text{NOW}}$	NAÏVE	HIST	SMART$^{\text{NOW}}$	NAÏVE	HIST	SMART$^{\text{NOW}}$	NAÏVE	HIST
$N = 200$ $\quad \lvert\mathcal{S}\rvert = 8.38 \cdot 10^{211}$						SMART completes in 108 sec using 284MB			
2	119	-24%	-13%	286	$+3\%$	$+45\%$	197	$+1\%$	$+53\%$
4	139	-27%	-15%	286	$+11\%$	$+51\%$	127	$+61\%$	$+58\%$
8	182	-32%	-24%	286	$+129\%$	$+62\%$	69	$+239\%$	$+62\%$
$N = 300$ $\quad \lvert\mathcal{S}\rvert = 8.38 \cdot 10^{211}$						SMART does not complete in 5 hrs using 512MB			
2	$^{s}552$	$^{s}+5\%$	$^{s}-5\%$	962	$+25\%$	$+11\%$	562	$+8\%$	$+7\%$
4	$^{d}490$	> 5hrs	$^{d}-16\%$	962	-	$+34\%$	352	-	$+12\%$
8	564	> 5hrs	-39%	962	-	$+50\%$	252	-	$+23\%$

- Both NAÏVE and HIST approaches outperform SMART in some cases

Slotted ring network protocol

- Protocol for local area networks : N is the number of nodes within the network ($K = N$, $|\mathcal{S}_k| = 10$ for all k, $|\mathcal{E}| = 3N$)

  E. Pastor, O. Roig, J. Cortadella, and R. Badia, *Petri net analysis using boolean manipulation*, ICATPN 1994

W	**Time** (sec)			**Total Memory** (MB)			**Max Memory** (MB)		
	SMART$^{\text{OW}}_{\text{N}}$	NAÏVE	HIST	SMART$^{\text{OW}}_{\text{N}}$	NAÏVE	HIST	SMART$^{\text{OW}}_{\text{N}}$	NAÏVE	HIST
$N = 200$ $\quad \lvert\mathcal{S}\rvert = 8.38 \cdot 10^{211}$							SMART completes in 108 sec using 284MB		
2	119	-24%	-13%	286	$+3\%$	$+45\%$	197	$+1\%$	$+53\%$
4	139	-27%	-15%	286	$+11\%$	$+51\%$	127	$+61\%$	$+58\%$
8	182	-32%	-24%	286	$+129\%$	$+62\%$	69	$+239\%$	$+62\%$
$N = 300$ $\quad \lvert\mathcal{S}\rvert = 8.38 \cdot 10^{211}$							SMART does not complete in 5 hrs using 512MB		
2	$^{s}552$	$^{s}+5\%$	$^{s}-5\%$	962	$+25\%$	$+11\%$	562	$+8\%$	$+7\%$
4	$^{d}490$	> 5hrs	$^{d}-16\%$	962	-	$+34\%$	352	-	$+12\%$
8	564	> 5hrs	-39%	962	-	$+50\%$	252	-	$+23\%$

- The memory overhead is not trivial
- Overall idle time increases $\Rightarrow$ overall memory consumption increases

Slotted ring network protocol

- Protocol for local area networks : N is the number of nodes within the network ($K = N$, $|\mathcal{S}_k| = 10$ for all k, $|\mathcal{E}| = 3N$)

  E. Pastor, O. Roig, J. Cortadella, and R. Badia, *Petri net analysis using boolean manipulation*, ICATPN 1994

W	**Time** (sec)			**Total Memory** (MB)			**Max Memory** (MB)		
	SMART$^{\text{NOW}}$	NAÏVE	HIST	SMART$^{\text{NOW}}$	NAÏVE	HIST	SMART$^{\text{NOW}}$	NAÏVE	HIST
$N = 200$ $\quad \lvert\mathcal{S}\rvert = 8.38 \cdot 10^{211}$							SMART completes in 108 sec using 284MB		
2	119	-24%	-13%	286	$+3\%$	$+45\%$	197	$+1\%$	$+53\%$
4	139	-27%	-15%	286	$+11\%$	$+51\%$	127	$+61\%$	$+58\%$
8	182	-32%	-24%	286	$+129\%$	$+62\%$	69	$+239\%$	$+62\%$
$N = 300$ $\quad \lvert\mathcal{S}\rvert = 8.38 \cdot 10^{211}$							SMART does not complete in 5 hrs using 512MB		
2	$^{s}552$	$^{s}+5\%$	$^{s}-5\%$	962	$+25\%$	$+11\%$	562	$+8\%$	$+7\%$
4	$^{d}490$	> 5hrs	$^{d}-16\%$	962	-	$+34\%$	352	-	$+12\%$
8	564	> 5hrs	-39%	962	-	$+50\%$	252	-	$+23\%$

- The excessive memory requirement could forces to trigger the dynamic load balancing or memory swapping earlier
- The HIST approach releases the heavy memory overhead in the NAÏVE one

Flexible manufacturing system

- System with three machines to process three different types of parts : N is the number of each type of parts

  ($K = 19$, $|\mathcal{S}_k| = N + 1$ for all k except $|\mathcal{S}_{17}| = 4$, $|\mathcal{S}_{12}| = 3$, and $|\mathcal{S}_7| = 2$, $|\mathcal{E}| = 20$)

  A. Miner and G. Ciardo, *Efficient reachability set generation and storage using DDs*, ICATPN 1999

W	**Time** (sec)			**Total Memory** (MB)			**Max Memory** (MB)		
	SMART[NOW]	NAÏVE	HIST	SMART[NOW]	NAÏVE	HIST	SMART[NOW]	NAÏVE	HIST
$N = 300$ $\quad \lvert\mathcal{S}\rvert = 3.64 \cdot 10^{27}$							SMART completes in 55 sec using 241MB		
2	79	-8%	-8%	243	$+12\%$	$+24\%$	121	$+26\%$	$+52\%$
4	91	[d]$+67\%$	-9%	243	$+102\%$	$+30\%$	119	$+205\%$	$+50\%$
8	260	> 5hrs	-30%	243	-	$+42\%$	103	-	$+47\%$
$N = 450$ $\quad \lvert\mathcal{S}\rvert = 6.90 \cdot 10^{29}$							SMART does not complete in 5 hrs using 512MB		
2	[s]257	[s]$+12\%$	[s]-14%	826	$+16\%$	$+5\%$	512	$+15\%$	$+7\%$
4	[d]311	> 5hrs	[d]-18%	826	-	$+33\%$	372	-	$+6\%$
8	959	> 5hrs	-25%	826	-	$+61\%$	343	-	$+6\%$

- It is a different type of model in comparison with the previous model

Flexible manufacturing system

- System with three machines to process three different types of parts : N is the number of each type of parts

  ($K = 19$, $|\mathcal{S}_k| = N + 1$ for all k except $|\mathcal{S}_{17}| = 4$, $|\mathcal{S}_{12}| = 3$, and $|\mathcal{S}_7| = 2$, $|\mathcal{E}| = 20$)

  A. Miner and G. Ciardo, *Efficient reachability set generation and storage using DDs*, ICATPN 1999

W	**Time** (sec)			**Total Memory** (MB)			**Max Memory** (MB)		
	SMART$^{QW}_{N}$	NAÏVE	HIST	SMART$^{QW}_{N}$	NAÏVE	HIST	SMART$^{QW}_{N}$	NAÏVE	HIST
$N = 300$ $\quad \lvert\mathcal{S}\rvert = 3.64 \cdot 10^{27}$							SMART completes in 55 sec using 241MB		
2	79	-8%	-8%	243	$+12\%$	$+24\%$	121	$+26\%$	$+52\%$
4	91	$^{d}+67\%$	-9%	243	$+102\%$	$+30\%$	119	$+205\%$	$+50\%$
8	260	$>$ 5hrs	-30%	243	-	$+42\%$	103	-	$+47\%$
$N = 450$ $\quad \lvert\mathcal{S}\rvert = 6.90 \cdot 10^{29}$							SMART does not complete in 5 hrs using 512MB		
2	$^{s}257$	$^{s}+12\%$	$^{s}-14\%$	826	$+16\%$	$+5\%$	512	$+15\%$	$+7\%$
4	$^{d}311$	$>$ 5hrs	$^{d}-18\%$	826	-	$+33\%$	372	-	$+6\%$
8	959	$>$ 5hrs	-25%	826	-	$+61\%$	343	-	$+6\%$

- The NAÏVE approach only outperforms SMART$^{QW}_{N}$ in some small case

Flexible manufacturing system

- System with three machines to process three different types of parts : N is the number of each type of parts

  ($K = 19$, $|\mathcal{S}_k| = N + 1$ for all k except $|\mathcal{S}_{17}| = 4$, $|\mathcal{S}_{12}| = 3$, and $|\mathcal{S}_7| = 2$, $|\mathcal{E}| = 20$)

  A. Miner and G. Ciardo, *Efficient reachability set generation and storage using DDs*, ICATPN 1999

W	**Time** (sec)			**Total Memory** (MB)			**Max Memory** (MB)		
	SMART[N]OW	NAÏVE	HIST	SMART[N]OW	NAÏVE	HIST	SMART[N]OW	NAÏVE	HIST
$N = 300$ $\quad \lvert\mathcal{S}\rvert = 3.64 \cdot 10^{27}$							SMART completes in 55 sec using 241MB		
2	79	-8%	-8%	243	$+12\%$	$+24\%$	121	$+26\%$	$+52\%$
4	91	[d]$+67\%$	-9%	243	$+102\%$	$+30\%$	119	$+205\%$	$+50\%$
8	260	$>$ 5hrs	-30%	243	-	$+42\%$	103	-	$+47\%$
$N = 450$ $\quad \lvert\mathcal{S}\rvert = 6.90 \cdot 10^{29}$							SMART does not complete in 5 hrs using 512MB		
2	[s]257	[s]$+12\%$	[s]-14%	826	$+16\%$	$+5\%$	512	$+15\%$	$+7\%$
4	[d]311	$>$ 5hrs	[d]-18%	826	-	$+33\%$	372	-	$+6\%$
8	959	$>$ 5hrs	-25%	826	-	$+61\%$	343	-	$+6\%$

- The HIST approach outperforms SMART[N]OW and controls memory usage well
- HIST approach works on various types of models

Round robin mutex protocol

- Round robin version of mutual exclusion algorithm : N is the number of processors involved

  ($K = N + 1$, $|\mathcal{S}_k| = 10$ for all k except $|\mathcal{S}_1| = N + 1$, $|\mathcal{E}| = 5N$).

  S. Graf, B. Steffen, and G. Lüttgen, *Compositional minimisation of finite state systems using interface specification*, Formal Aspect of Computer 1996

W	**Time** (sec)			**Total Memory** (MB)			**Max Memory** (MB)		
	SMART$^{\text{OW}}_{\text{N}}$	NAÏVE	HIST	SMART$^{\text{OW}}_{\text{N}}$	NAÏVE	HIST	SMART$^{\text{OW}}_{\text{N}}$	NAÏVE	HIST
$N = 800$ $\quad \lvert\mathcal{S}\rvert = 1.20 \cdot 10^{196}$							SMART completes in 27 sec using 290MB		
2	29	+37%	+6%	293	+110%	+85%	215	+52%	+63%
4	36	+33%	+8%	293	+348%	+109%	130	+186%	+65%
8	51	+33%	+5%	293	+807%	+148%	73	+433%	+73%
$N = 1100$ $\quad \lvert\mathcal{S}\rvert = 3.36 \cdot 10^{334}$							SMART does not complete in 5 hrs using 512MB		
2	^{d}65	^{s}+62%	^{s}+18%	794	+46%	+6%	379	+79%	+30%
4	47	^{s}+131%	^{d}+10%	794	+119%	+38%	265	+104%	+40%
8	56	^{d}+164%	+7%	794	+299%	+50%	173	+126%	+38%

- The NAÏVE approach performs excessive fire prediction but most speculative firing results are not useful

Round robin mutex protocol

- Round robin version of mutual exclusion algorithm : N is the number of processors involved

  ($K = N + 1$, $|\mathcal{S}_k| = 10$ for all k except $|\mathcal{S}_1| = N + 1$, $|\mathcal{E}| = 5N$).

  S. Graf, B. Steffen, and G. Lüttgen, *Compositional minimisation of finite state systems using interface specification*, Formal Aspect of Computer 1996

W	**Time** (sec)			**Total Memory** (MB)			**Max Memory** (MB)		
	SMART$^{\text{OW}}_{\text{N}}$	NAÏVE	HIST	SMART$^{\text{OW}}_{\text{N}}$	NAÏVE	HIST	SMART$^{\text{OW}}_{\text{N}}$	NAÏVE	HIST
$N = 800$ $\quad \lvert\mathcal{S}\rvert = 1.20 \cdot 10^{196}$							SMART completes in 27 sec using 290MB		
2	29	+37%	+6%	293	+110%	+85%	215	+52%	+63%
4	36	+33%	+8%	293	+348%	+109%	130	+186%	+65%
8	51	+33%	+5%	293	+807%	+148%	73	+433%	+73%
$N = 1100$ $\quad \lvert\mathcal{S}\rvert = 3.36 \cdot 10^{334}$							SMART does not complete in 5 hrs using 512MB		
2	^{d}65	^{s}+62%	^{s}+18%	794	+46%	+6%	379	+79%	+30%
4	47	^{s}+131%	^{d}+10%	794	+119%	+38%	265	+104%	+40%
8	56	^{d}+164%	+7%	794	+299%	+50%	173	+126%	+38%

- The HIST approach controls the space usage well by not to perform prediction if no firing pattern exists

Runway monitoring system

- Avionics system : monitors T targets with S speeds on a $X \times Y \times Z$ runway ($K = 5(T+1)$, $|\mathcal{S}_{5+5i}| = 3$, $|\mathcal{S}_{4+5i}| = 14$, $|\mathcal{S}_{3+5i}| = 1+X(10+6(S-1))$, $|\mathcal{S}_{2+5i}| = 1+Y(10+6(S-1))$, $|\mathcal{S}_{1+5i}| = 1+Z(10+6(S-1))$, for $i = 0, ..., T$, except $|\mathcal{S}_{4+5T}| = 7$, $|\mathcal{E}| = 49 + T(56 + (Y-2)(31 + (X-2)(13+4Z)) + 3(X-2)(1+YZ) + 2X + 5Y + 3Z))$.

  R. Siminiceanu and G. Ciardo, *Formal verification of the NASA Runway Safety Monitor*, AVoCS 2004

W	**Time** (sec)			**Total Memory** (MB)			**Max Memory** (MB)		
	SMART$^{\text{QW}}$	NAÏVE	HIST	SMART$^{\text{QW}}$	NAÏVE	HIST	SMART$^{\text{QW}}$	NAÏVE	HIST
$Z = 2$ $\quad \lvert\mathcal{S}\rvert = 1.51 \cdot 10^{15}$							SMART completes in 236 sec using 314MB		
2	731	> 10hrs	-2%	332	-	$+39\%$	191	-	$+48\%$
4	938	> 10hrs	-8%	332	-	$+88\%$	190	-	$+30\%$
8	1480	> 10hrs	-22%	332	-	$+128\%$	173	-	$+13\%$
$Z = 3$ $\quad \lvert\mathcal{S}\rvert = 5.07 \cdot 10^{15}$							SMART does not complete in 10 hrs using 512MB		
2	$^{s}11280$	> 10hrs	$^{s}-1\%$	962	-	$+10\%$	595	-	$+16\%$
4	$^{d}9762$	> 10hrs	$^{d}-15\%$	962	-	$+31\%$	371	-	$+8\%$
8	$^{d}14101$	> 10hrs	$^{d}-17\%$	962	-	$+58\%$	359	-	$+6\%$

- The HIST approach does work in practice

Conclusions

Conclusions and future work

- **Conclusions** :
  - Pattern recognition approach is capable to guide the speculative firing prediction to improve the runtime of SMARTNOW.
  - This approach does work on some real world model.
  - It seems applicable to the breadth-first iteration.
- **Future research directions** :
  - Explore more sophisticated, but still low-overhead, heuristics to improve the usefulness of the predicted events.
  - Augment this approach to include information about the current memory consumption.

Thank You!