Verifying the Concentration Property of Permutation Networks by BDDs

Tripti Jain and Klaus Schneider

Embedded Systems Chair
Department of Computer Science
University of Kaiserslautern,Germany

MEMOCODE 2016, November 18-20, 2016

Table of Contents

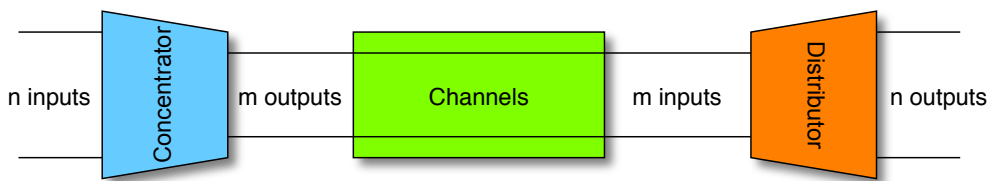
Outline

Concentrator – Definition

- **Pinsker (1973):**

  a concentrator is a circuit with n inputs and $m \leq n$ outputs that can route any given number $k \leq m$ of valid inputs to some k of its m outputs

- theoretical size $O(n)$
- theoretical depth $O(\log(n))$
- applications: interconnection networks, communication lines etc.



Concentrator – Practical Implementations

- **M. Chien and A. Oruç (1994):**
  - sorter as concentrators
  - $O(\log(n)^2)$ depth
  - $O(n \log(n))$ size (in terms of comparators)
- **Narasimha (1994):**
  - permutation networks as concentrators
    (reverse banyan flip shuffle permutation network with butterfly)
  - $O(n)$ depth
  - $O(n \log(n))$ size

- Which other permutation networks could be used as concentrators?
- Are there some that can be configured faster, e.g. in time $O(\log(n)^2)$?

Outline

Permutation Networks

- basic element: 2×2 crossbar switch

$$\begin{cases} y_0 = (s \Rightarrow x_1 \mid x_0) \\ y_1 = (s \Rightarrow x_0 \mid x_1) \end{cases}$$

Permutation Networks

- basic element: 2×2 crossbar switch

$$\begin{cases} y_0 = (s \Rightarrow x_1 \mid x_0) \\ y_1 = (s \Rightarrow x_0 \mid x_1) \end{cases}$$

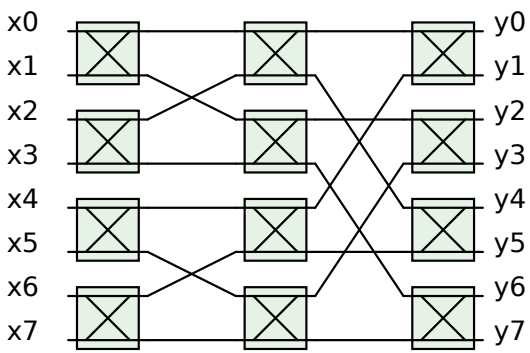
- switches are organized in a grid with $\frac{n}{2}$ rows and $\log_2(n)$ columns

Permutation Networks

- basic element: 2×2 crossbar switch

$$\begin{cases} y_0 = (s \Rightarrow x_1 \mid x_0) \\ y_1 = (s \Rightarrow x_0 \mid x_1) \end{cases}$$

- switches are organized in a grid with $\frac{n}{2}$ rows and $\log_2(n)$ columns

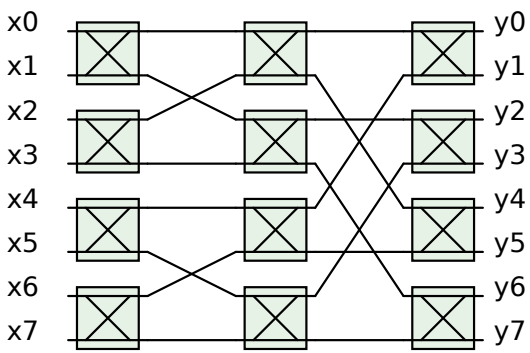


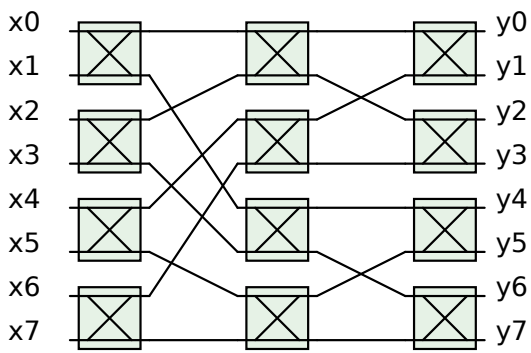
Permutation Networks

- basic element: 2×2 crossbar switch

$$\begin{cases} y_0 = (s \Rightarrow x_1 \mid x_0) \\ y_1 = (s \Rightarrow x_0 \mid x_1) \end{cases}$$

- switches are organized in a grid with $\frac{n}{2}$ rows and $\log_2(n)$ columns





Topology of the Networks

depends on two parameters:

- possibly partitioning columns into blocks

Topology of the Networks

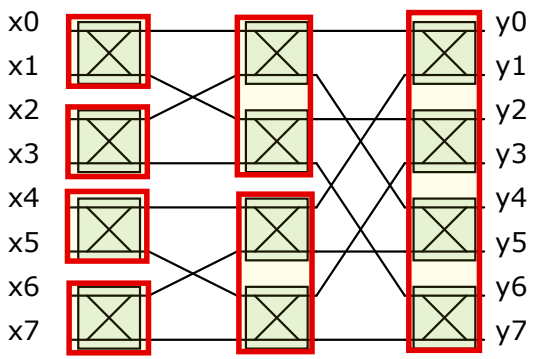
depends on two parameters:

- possibly partitioning columns into blocks
  - banyan network: number of blocks doubles from right to left
  - reverse banyan network: number of blocks doubles from left to right
  - multistage network: no partitioning

Topology of the Networks

depends on two parameters:

- possibly partitioning columns into blocks
  - banyan network: number of blocks doubles from right to left
  - reverse banyan network: number of blocks doubles from left to right
  - multistage network: no partitioning



Topology of the Networks

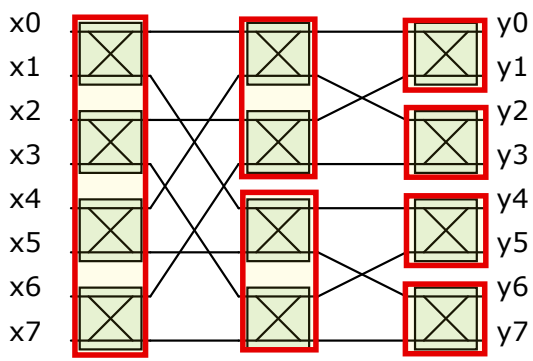
depends on two parameters:

- possibly partitioning columns into blocks
  - banyan network: number of blocks doubles from right to left
  - reverse banyan network: number of blocks doubles from left to right
  - multistage network: no partitioning

Topology of the Networks

depends on two parameters:

- possibly partitioning columns into blocks
  - banyan network: number of blocks doubles from right to left
  - reverse banyan network: number of blocks doubles from left to right
  - multistage network: no partitioning



Topology of the Networks

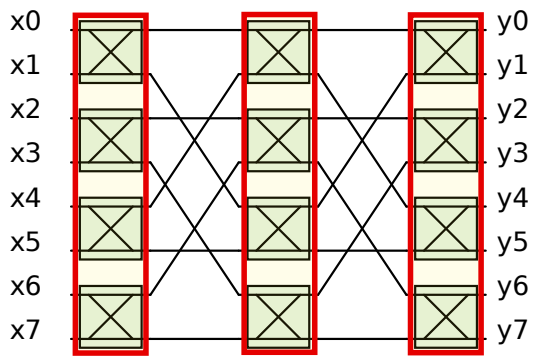
depends on two parameters:

- possibly partitioning columns into blocks
  - banyan network: number of blocks doubles from right to left
  - reverse banyan network: number of blocks doubles from left to right
  - multistage network: no partitioning

Topology of the Networks

depends on two parameters:

- possibly partitioning columns into blocks
  - banyan network: number of blocks doubles from right to left
  - reverse banyan network: number of blocks doubles from left to right
  - multistage network: no partitioning



Topology of the Networks

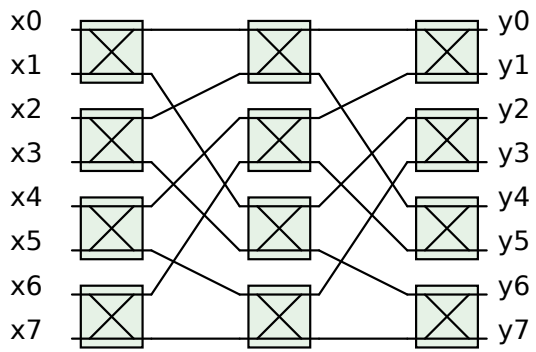
depends on two parameters:

- possibly partitioning columns into blocks
  - banyan network: number of blocks doubles from right to left
  - reverse banyan network: number of blocks doubles from left to right
  - multistage network: no partitioning
- specific permutation between the columns

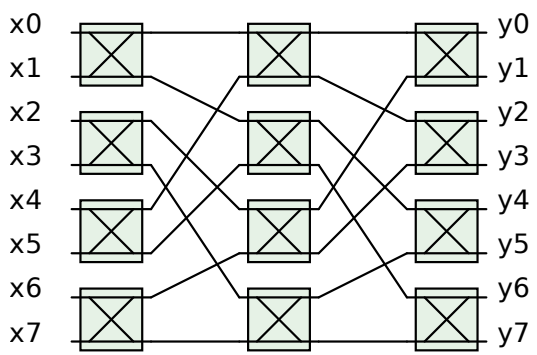
Topology of the Networks

depends on two parameters:

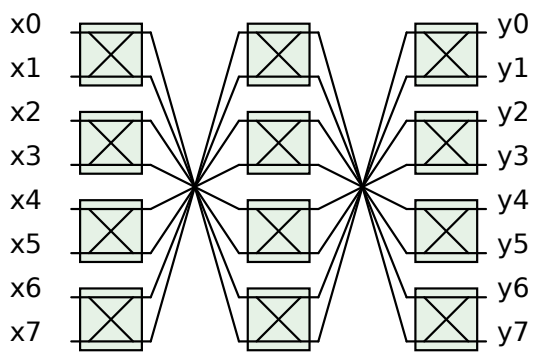
- possibly partitioning columns into blocks
  - banyan network: number of blocks doubles from right to left
  - reverse banyan network: number of blocks doubles from left to right
  - multistage network: no partitioning
- specific permutation between the columns



MS-FS



MS-PS



MS-MR

Permutations

popular permutations for n=8:

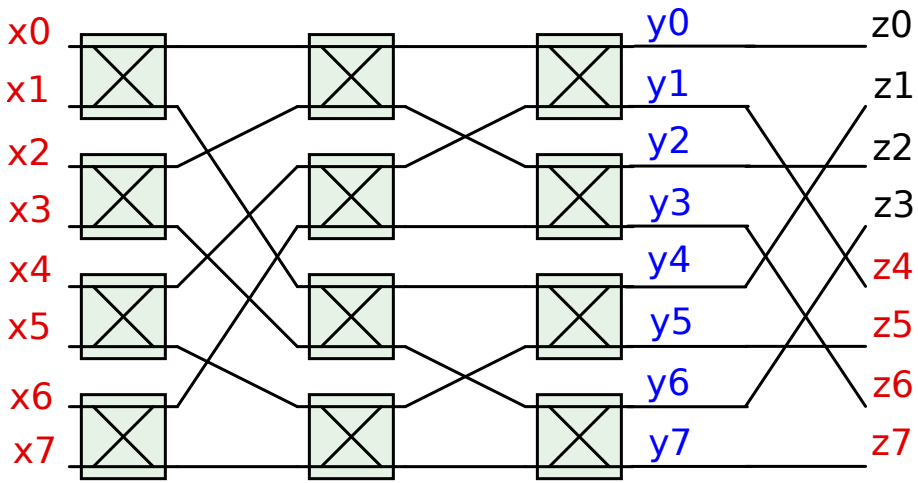
Permutation	0	1	2	3	4	5	6	7
PS := PerfectShuffle	0	2	4	6	1	3	5	7
FS := FlipShuffle	0	4	1	5	2	6	3	7
BF := Butterfly	0	4	2	6	1	5	3	7
MR := Mirror	7	6	5	4	3	2	1	0
RM := RevMirror	7	3	5	1	6	2	4	0
OP := OutputPerm	7	0	5	2	3	4	1	6

Permutations

considering 3×6 permutation networks

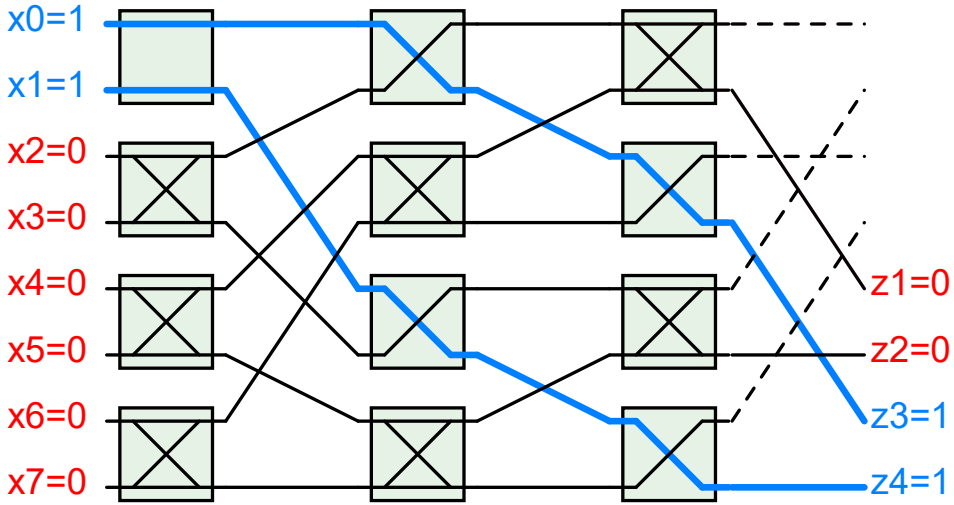
	MS	RB	BY
PS			
FS			
BF			
MR			
RM			
OP			

Narasimha's Concentrator



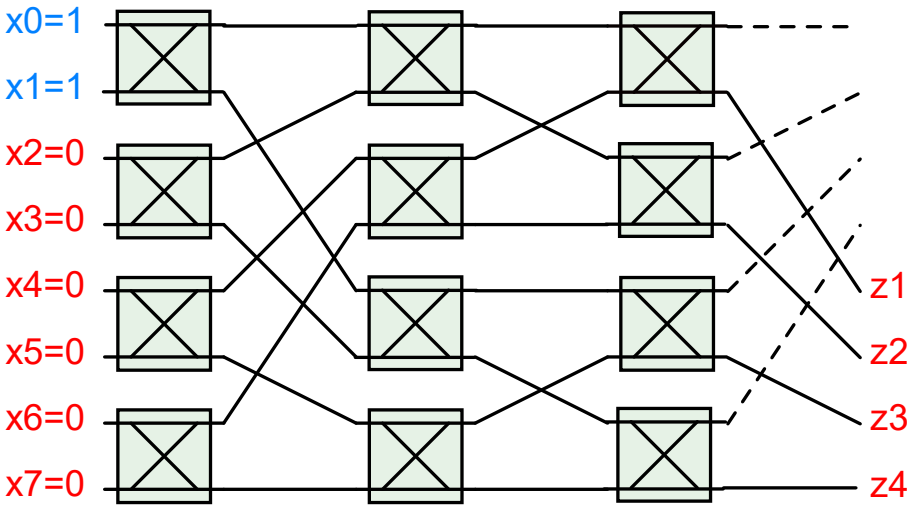
- RB-FS-BF permutation network
- n=8
- input binary vector: $\vec{x} = (x_0, \ldots, x_7)$
- output binary vector: $\vec{z} = (z_4, \ldots, z_7)$
- permutated output: $\vec{y} = (y_0, \ldots, y_7)$

Example 1



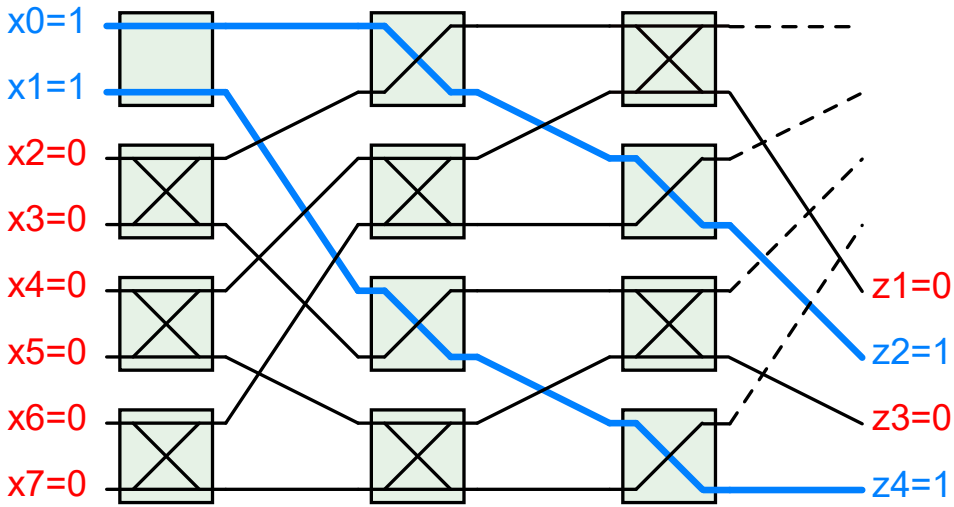
- RB-FS-BF permutation network
- (8,4)-concentrator: n=8, m=4
- k=2
- input binary vector =(1,1,0,0,0,0,0,0)
- output binary vector = (0,0,1,1)

Example 2



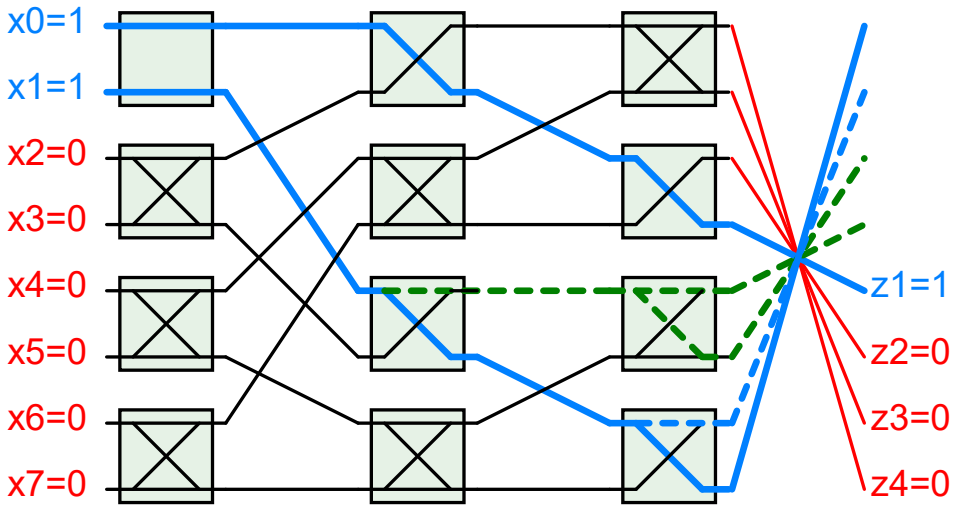
- RB-FS-FS permutation network

Example 2



- RB-FS-FS permutation network
- (8,4)-concentrator: n=8, m=4
- k=2
- input binary vector =(1,1,0,0,0,0,0,0)
- output binary vector = (0,1,0,1)

Example 3



- RB-FS-MR permutation network
- n=8, m=4
- k=2
- input binary vector =(1,1,0,0,0,0,0,0)
- output binary vector = (1,0,0,0)
- it is not an (8,4)-concentrator

Outline

Analysis of Permutation Networks

- network to boolean equations

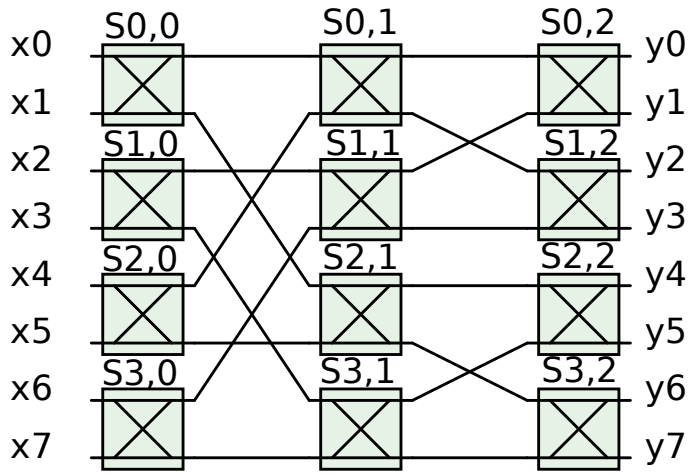
$$\vec{y} = \mathrm{Net}_f(\vec{s}, \vec{x})$$

  - $\vec{x}$ - input variables
  - $\vec{s}$ - switch configuration
  - $\vec{y}$ - output variables
- possible permutations

$$\mathrm{Perm}_f(\vec{x}, \vec{y}) :\Leftrightarrow \exists \vec{s}.\ \vec{y} = \mathrm{Net}_f(\vec{s}, \vec{x})$$

  - existential quantification over the variables $\vec{s}$
  - $\mathrm{Perm}_f(\vec{x}, \vec{y})$ - permutation predicate for network f
  - $\mathrm{Perm}_f(\vec{x}, \vec{y})$ relates those (x,y) where x can be routed to y

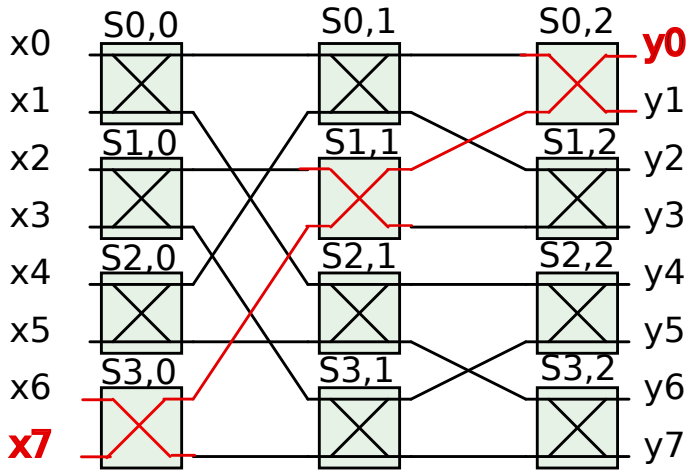
Example



Reverse Banyan Butterfly ($RB - BF$)

$$y_0 := (s_{0,2} \Rightarrow (s_{1,1} \Rightarrow (s_{3,0} \Rightarrow x_7 \mid x_6) \mid (s_{1,0} \Rightarrow x_3 \mid x_2)) \mid (s_{0,1} \Rightarrow (s_{2,0} \Rightarrow x_5 \mid x_4) \mid (s_{0,0} \Rightarrow x_1 \mid x_0)))$$
$$y_1 := (s_{0,2} \Rightarrow (s_{0,1} \Rightarrow (s_{2,0} \Rightarrow x_5 \mid x_4) \mid (s_{0,0} \Rightarrow x_1 \mid x_0)) \mid (s_{1,1} \Rightarrow (s_{3,0} \Rightarrow x_7 \mid x_6) \mid (s_{1,0} \Rightarrow x_3 \mid x_2)))$$
$$y_2 := (s_{1,2} \Rightarrow (s_{1,1} \Rightarrow (s_{1,0} \Rightarrow x_3 \mid x_2) \mid (s_{3,0} \Rightarrow x_7 \mid x_6)) \mid (s_{0,1} \Rightarrow (s_{0,0} \Rightarrow x_1 \mid x_0) \mid (s_{2,0} \Rightarrow x_5 \mid x_4)))$$
$$y_3 := (s_{1,2} \Rightarrow (s_{0,1} \Rightarrow (s_{0,0} \Rightarrow x_1 \mid x_0) \mid (s_{2,0} \Rightarrow x_5 \mid x_4)) \mid (s_{1,1} \Rightarrow (s_{1,0} \Rightarrow x_3 \mid x_2) \mid (s_{3,0} \Rightarrow x_7 \mid x_6)))$$
$$y_4 := (s_{2,2} \Rightarrow (s_{3,1} \Rightarrow (s_{3,0} \Rightarrow x_6 \mid x_7) \mid (s_{1,0} \Rightarrow x_2 \mid x_3)) \mid (s_{2,1} \Rightarrow (s_{2,0} \Rightarrow x_4 \mid x_5) \mid (s_{0,0} \Rightarrow x_0 \mid x_1)))$$
$$y_5 := (s_{2,2} \Rightarrow (s_{2,1} \Rightarrow (s_{2,0} \Rightarrow x_4 \mid x_5) \mid (s_{0,0} \Rightarrow x_0 \mid x_1)) \mid (s_{3,1} \Rightarrow (s_{3,0} \Rightarrow x_6 \mid x_7) \mid (s_{1,0} \Rightarrow x_2 \mid x_3)))$$
$$y_6 := (s_{3,2} \Rightarrow (s_{3,1} \Rightarrow (s_{1,0} \Rightarrow x_2 \mid x_3) \mid (s_{3,0} \Rightarrow x_6 \mid x_7)) \mid (s_{2,1} \Rightarrow (s_{0,0} \Rightarrow x_0 \mid x_1) \mid (s_{2,0} \Rightarrow x_4 \mid x_5)))$$
$$y_7 := (s_{3,2} \Rightarrow (s_{2,1} \Rightarrow (s_{0,0} \Rightarrow x_0 \mid x_1) \mid (s_{2,0} \Rightarrow x_4 \mid x_5)) \mid (s_{3,1} \Rightarrow (s_{1,0} \Rightarrow x_2 \mid x_3) \mid (s_{3,0} \Rightarrow x_6 \mid x_7)))$$

Example



Reverse Banyan Butterfly (RB − BF)

$$y_0 := (s_{0,2} \Rightarrow (s_{1,1} \Rightarrow (s_{3,0} \Rightarrow x_7 \mid x_6) \mid (s_{1,0} \Rightarrow x_3 \mid x_2)) \mid (s_{0,1} \Rightarrow (s_{2,0} \Rightarrow x_5 \mid x_4) \mid (s_{0,0} \Rightarrow x_1 \mid x_0)))$$
$$y_1 := (s_{0,2} \Rightarrow (s_{0,1} \Rightarrow (s_{2,0} \Rightarrow x_5 \mid x_4) \mid (s_{0,0} \Rightarrow x_1 \mid x_0)) \mid (s_{1,1} \Rightarrow (s_{3,0} \Rightarrow x_7 \mid x_6) \mid (s_{1,0} \Rightarrow x_3 \mid x_2)))$$
$$y_2 := (s_{1,2} \Rightarrow (s_{1,1} \Rightarrow (s_{1,0} \Rightarrow x_3 \mid x_2) \mid (s_{3,0} \Rightarrow x_7 \mid x_6)) \mid (s_{0,1} \Rightarrow (s_{0,0} \Rightarrow x_1 \mid x_0) \mid (s_{2,0} \Rightarrow x_5 \mid x_4)))$$
$$y_3 := (s_{1,2} \Rightarrow (s_{0,1} \Rightarrow (s_{0,0} \Rightarrow x_1 \mid x_0) \mid (s_{2,0} \Rightarrow x_5 \mid x_4)) \mid (s_{1,1} \Rightarrow (s_{1,0} \Rightarrow x_3 \mid x_2) \mid (s_{3,0} \Rightarrow x_7 \mid x_6)))$$
$$y_4 := (s_{2,2} \Rightarrow (s_{3,1} \Rightarrow (s_{3,0} \Rightarrow x_6 \mid x_7) \mid (s_{1,0} \Rightarrow x_2 \mid x_3)) \mid (s_{2,1} \Rightarrow (s_{2,0} \Rightarrow x_4 \mid x_5) \mid (s_{0,0} \Rightarrow x_0 \mid x_1)))$$
$$y_5 := (s_{2,2} \Rightarrow (s_{2,1} \Rightarrow (s_{2,0} \Rightarrow x_4 \mid x_5) \mid (s_{0,0} \Rightarrow x_0 \mid x_1)) \mid (s_{3,1} \Rightarrow (s_{3,0} \Rightarrow x_6 \mid x_7) \mid (s_{1,0} \Rightarrow x_2 \mid x_3)))$$
$$y_6 := (s_{3,2} \Rightarrow (s_{3,1} \Rightarrow (s_{1,0} \Rightarrow x_2 \mid x_3) \mid (s_{3,0} \Rightarrow x_6 \mid x_7)) \mid (s_{2,1} \Rightarrow (s_{0,0} \Rightarrow x_0 \mid x_1) \mid (s_{2,0} \Rightarrow x_4 \mid x_5)))$$
$$y_7 := (s_{3,2} \Rightarrow (s_{2,1} \Rightarrow (s_{0,0} \Rightarrow x_0 \mid x_1) \mid (s_{2,0} \Rightarrow x_4 \mid x_5)) \mid (s_{3,1} \Rightarrow (s_{1,0} \Rightarrow x_2 \mid x_3) \mid (s_{3,0} \Rightarrow x_6 \mid x_7)))$$

Analysis of Permutation Networks

- network to boolean equation

$$\vec{y} = \mathrm{Net}_f(\vec{s}, \vec{x})$$

  - $\vec{x}$ - inputs variables
  - $\vec{s}$ - switch configuration
  - $\vec{y}$ - output variables

- possible permutations

$$\mathrm{Perm}_f(\vec{x}, \vec{y}) :\Leftrightarrow \exists \vec{s}.\ \vec{y} = \mathrm{Net}_f(\vec{s}, \vec{x})$$

  - existential quantification over the variables $\vec{s}$
  - $\mathrm{Perm}_f(\vec{x}, \vec{y})$ - permutation predicate for network f
  - $\mathrm{Perm}_f(\vec{x}, \vec{y})$ relates those (x,y) where x can be routed to y

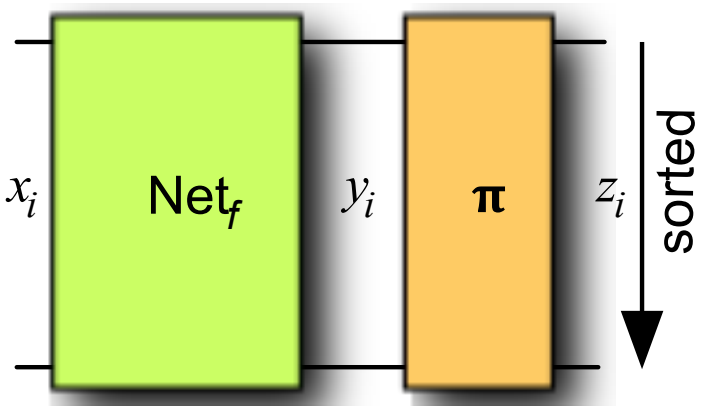
Experimental Results

	MS			RB			BY		
	ne	*np*	*sp*	*ne*	*np*	*sp*	*ne*	*np*	*sp*
PS	44799	200	11602	13117	152	4900	44799	191	11602
FS	45795	209	11602	45051	191	11602	17927	152	4900
BF	18179	160	4900	45051	200	11602	45543	209	11602
MR	2517	320	1296	2517	320	1296	2517	320	1296
RM	18179	160	4900	45051	200	11602	45543	209	11602
OP	18179	160	4900	45051	200	11602	45543	209	11602

- variable ordering $\vec{s} \preceq \vec{y} \preceq \vec{x}$
- ne: number of BDD nodes for $\vec{y} = \mathrm{Net}_f(\vec{s}, \vec{x})$
- np: number of BDD nodes for $\mathrm{Perm}_f(\vec{x}, \vec{y})$
- sp: number of satisfying assignments of $\mathrm{Perm}_f(\vec{x}, \vec{y})$

Outline

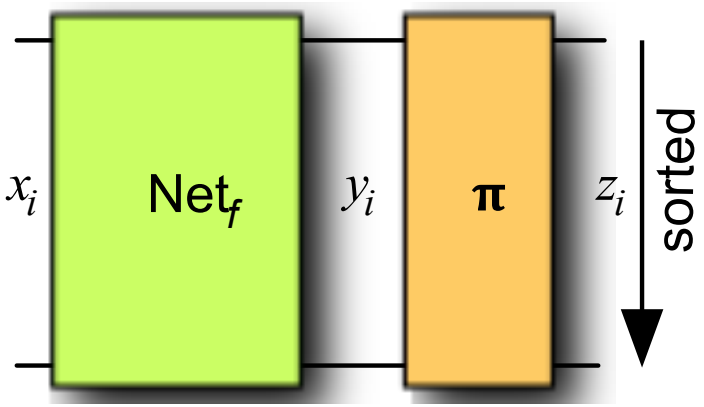
Sorter



for every input k-vector

- $\{z_0, \ldots, z_{n-k-1}\}$: contains the zeros
- $\{z_{n-k}, \ldots, z_{n-1}\}$: contains the ones

Sorter

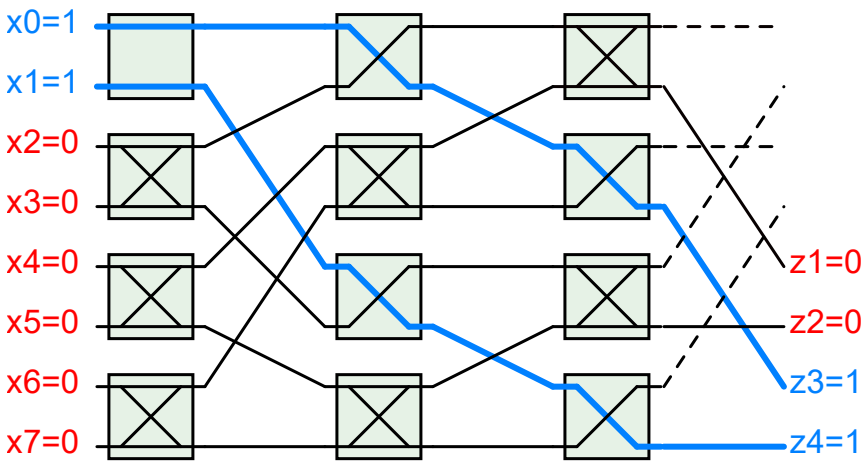


for every input k-vector

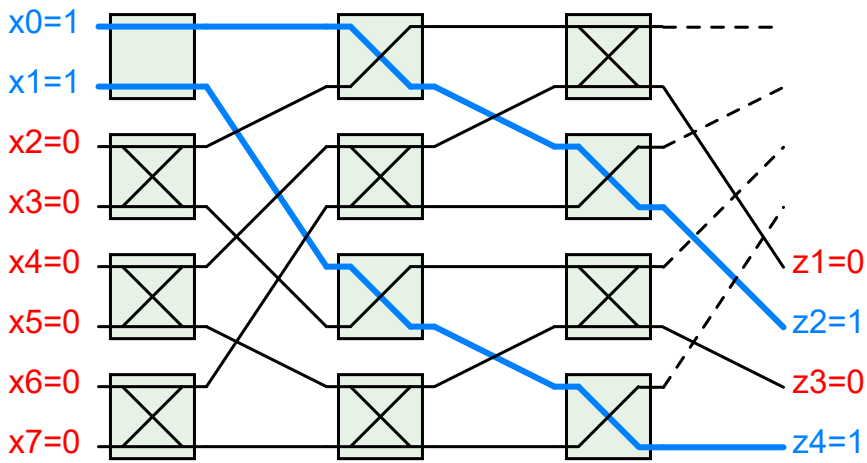
- $\{z_0, \ldots, z_{n-k-1}\}$: contains the zeros
- $\{z_{n-k}, \ldots, z_{n-1}\}$: contains the ones

$$\text{Sorted}(\vec{z}) :\Leftrightarrow \bigwedge_{i=0}^{n-2} \left(z_i \rightarrow \bigwedge_{j=i+1}^{n-1} z_j \right)$$

Example



RB-FS+BF
(sorter and concentrator)



RB-FS+FS
(concentrator, but not a sorter)

Sorting and Concentrating with a Given Permutation

- **sorting**

$$\mathsf{SortedInputs}_\pi(\vec{x}) :\Leftrightarrow \exists \vec{y} \exists \vec{z}.\ \mathsf{Perm}_f(\vec{x}, \vec{y}) \wedge \mathsf{Sorted}(\vec{z}) \wedge \vec{z} = \pi(\vec{y})$$

  - $\mathsf{SortedInputs}_\pi(\vec{x})$ holds if $\vec{x}$ can be sorted with output permutation π
  - must be true for all input vectors $\vec{x}$ with output permutation π

Sorting and Concentrating with a Given Permutation

- **sorting**

$$\mathsf{SortedInputs}_\pi(\vec{x}) :\Leftrightarrow \exists\vec{y}\exists\vec{z}.\ \mathsf{Perm}_f(\vec{x},\vec{y}) \wedge \mathsf{Sorted}(\vec{z}) \wedge \vec{z} = \pi(\vec{y})$$

  - $\mathsf{SortedInputs}_\pi(\vec{x})$ holds if $\vec{x}$ can be sorted with output permutation π
  - must be true for all input vectors $\vec{x}$ with output permutation π

- **concentrating**

$$\mathsf{CnctrInputs}^m_\pi(\vec{x}) :\Leftrightarrow \mathsf{MaxValid}(m,\vec{x}) \rightarrow \exists\vec{y}\exists\vec{z}.\ \mathsf{Perm}_f(\vec{x},\vec{y}) \wedge \neg \bigvee_{i=0}^{n-m-1} z_i$$

  - $\mathsf{MaxValid}(m,\vec{x})$ means: at most m of the entries of $\vec{x}$ are valid
  - $\mathsf{CnctrInputs}^m_\pi(\vec{x})$ holds if $\vec{x}$ can be routed to an output $\vec{z}$ such that the valid bits of $\vec{z}$ are in $\{z_{n-m},\ldots,z_{n-1}\}$
  - must be true for all input vectors $\vec{x}$ with output permutation π

Experimental Results

	ID	PS	FS	BF	MR	RM	OP
MS – PS	–	–	⊗	x	–	x	x
MS – FS	–	–	x	⊗	–	⊗	x
MS – BF	–	–	–	–	–	–	–
MS – MR	–	–	–	–	–	–	–
MS – RM	–	–	–	–	–	–	–
MS – OP	–	–	–	–	–	–	–
RB – PS	–	–	–	–	–	–	–
RB – FS	–	–	⊗	x	–	x	x
RB – BF	–	–	⊗	x	–	x	x
RB – MR	–	–	–	–	–	–	–
RB – RM	–	–	⊗	x	–	x	x
RB – OP	–	–	⊗	x	–	x	x
BY – PS	–	–	⊗	x	–	x	x
BY – FS	–	–	–	–	–	–	–
BY – BF	–	–	x	⊗	–	⊗	x
BY – MR	–	–	–	–	–	–	–
BY – RM	–	–	x	⊗	–	⊗	x
BY – OP	–	–	x	x	–	x	⊗

- ⊗: concentrator, but not sorter
- x: both sorter and concentrator

Checking for existence of output permutations

- so far, we checked whether a network like RB-FS can work as a sorter or concentrator with a particular output permutation like BF
- next, we want to check whether for a network like RB-FS there **exists an output permutation** that allows us to use it as a sorter or concentrator

BDDs for NumValidOutputs$(k, \vec{y})$

- computing BDDs for NumValidOutputs$(k, \vec{y})$ for $k = 1, \ldots, n-1$

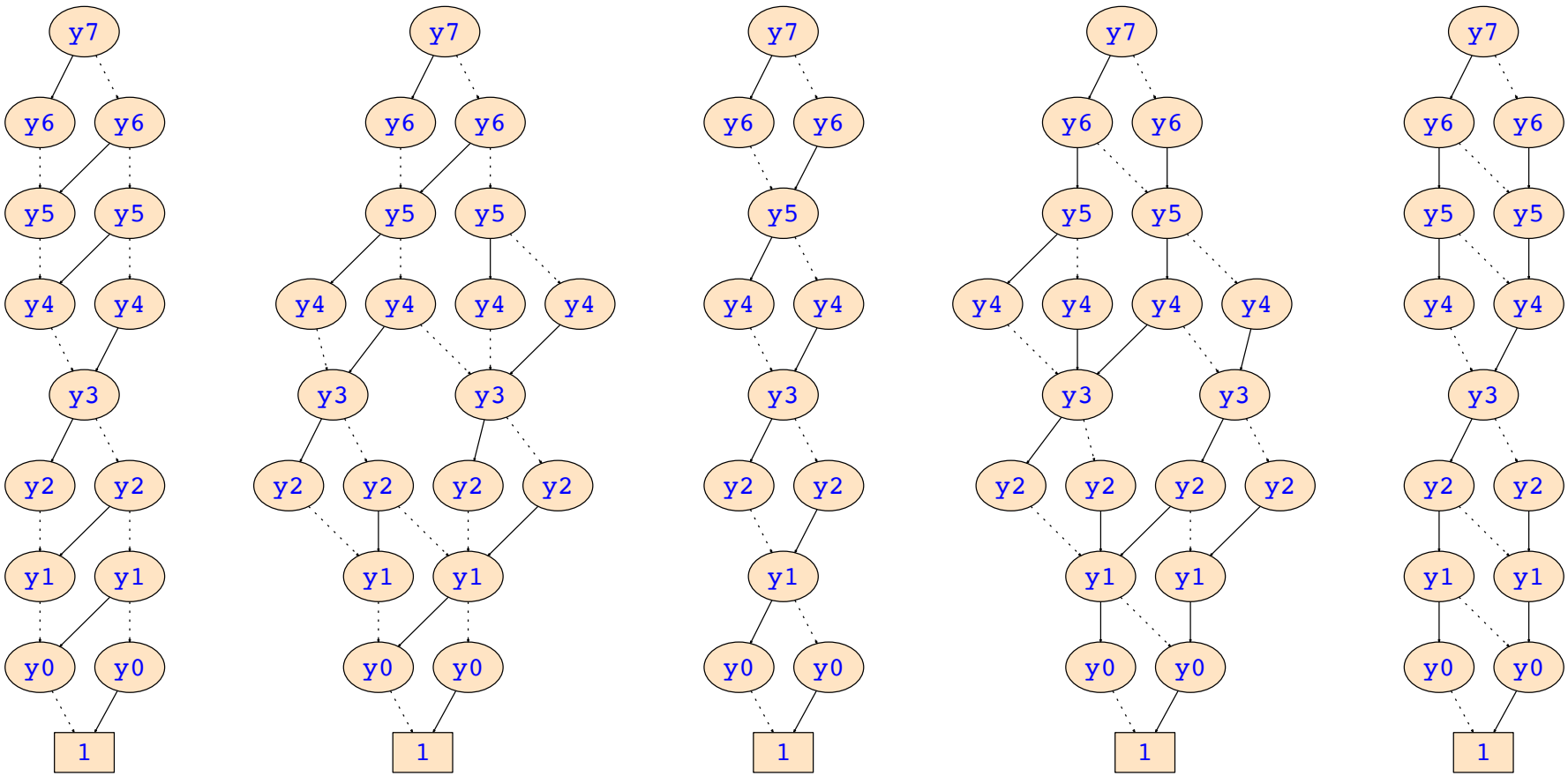
$$\text{NumValidOutputs}(k, \vec{y}) :\Leftrightarrow \forall \vec{x}.\text{NumValid}(k, \vec{x}) \rightarrow \text{Perm}_f(\vec{x}, \vec{y})$$

  - NumValid$(k, \vec{x})$ means k of the bits $x_0, \ldots, x_{n-1}$ are true
  - NumValidOutputs$(k, \vec{y})$ holds if all input vectors $\vec{x}$ with exactly k valid bits can be routed to y

BDDs for NumValidOutputs$(k, \vec{y})$

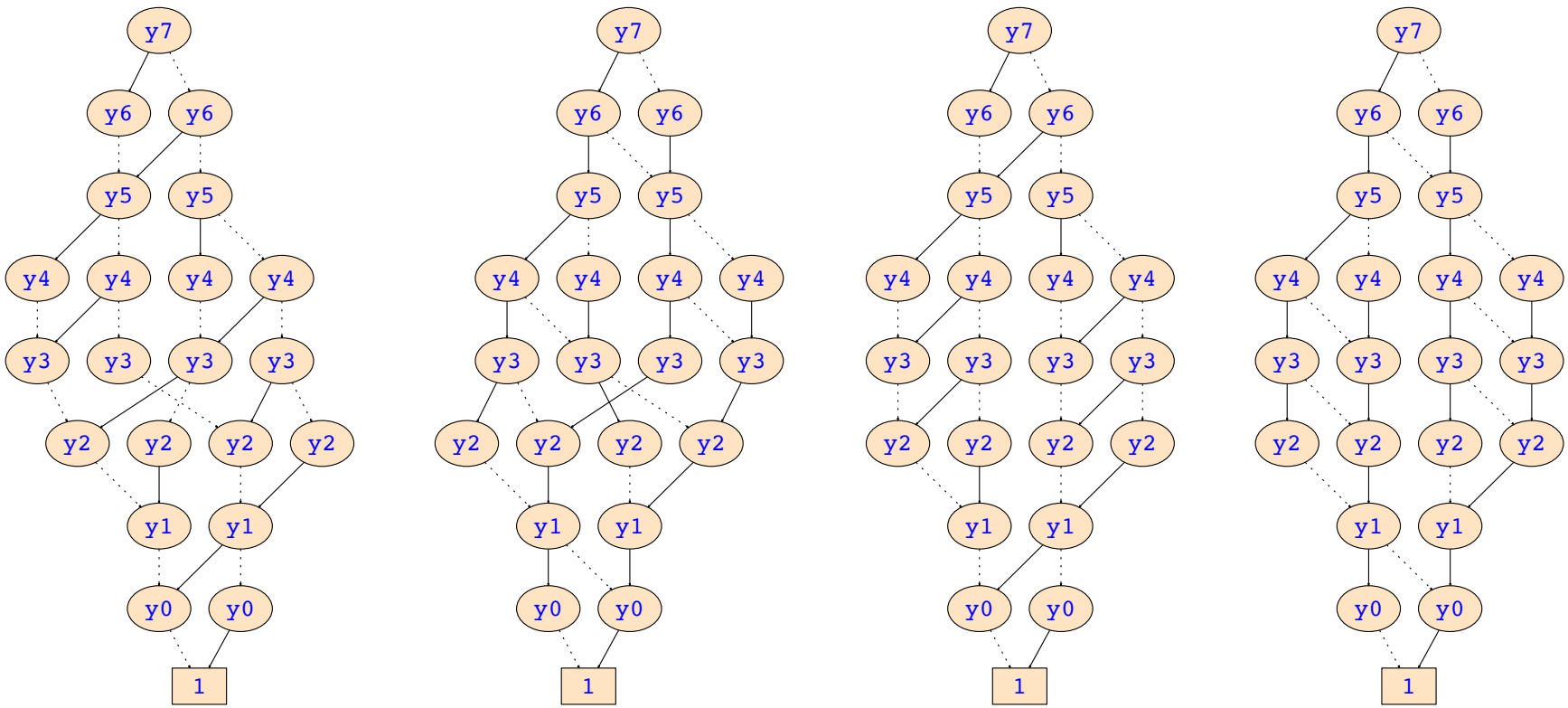
Strategy S1: MS − PS, RB − FS, RB − BF, RB − RM, RB − OP, BY − PS

2 3 4 5 6



BDDs for NumValidOutputs$(k, \vec{y})$

Strategy S2: MS − FS, BY − BF, BY − RM		Strategy S3: BY − OP	
2	6	2	6



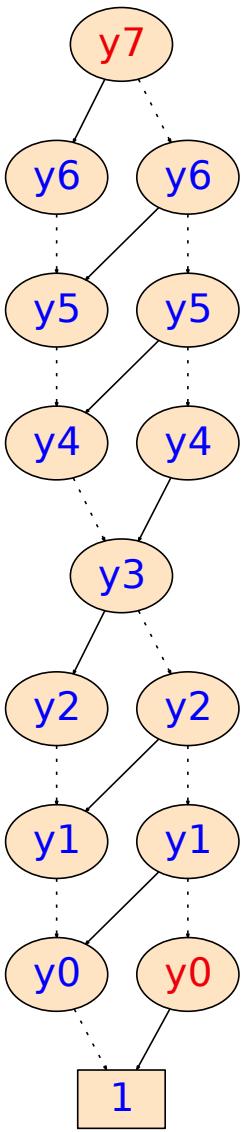
Determining Output Permutation

- computing BDDs for NumValidOutputs$(k, \vec{y})$ for $k = 1, \ldots, n-1$

$$\text{NumValidOutputs}(k, \vec{y}) :\Leftrightarrow \forall \vec{x}.\text{NumValid}(k, \vec{x}) \rightarrow \text{Perm}_f(\vec{x}, \vec{y})$$

- selecting an arbitrary model for NumValidOutputs$(1, \vec{y})$ where some variable $y_{i_{n-1}}$ is true as z_{n-1}
- check whether there is a model for NumValidOutputs$(2, \vec{y})$ where another variable $y_{i_{n-2}}$ is true in this BDD, if so, we can pick one of these variables as z_{n-2}

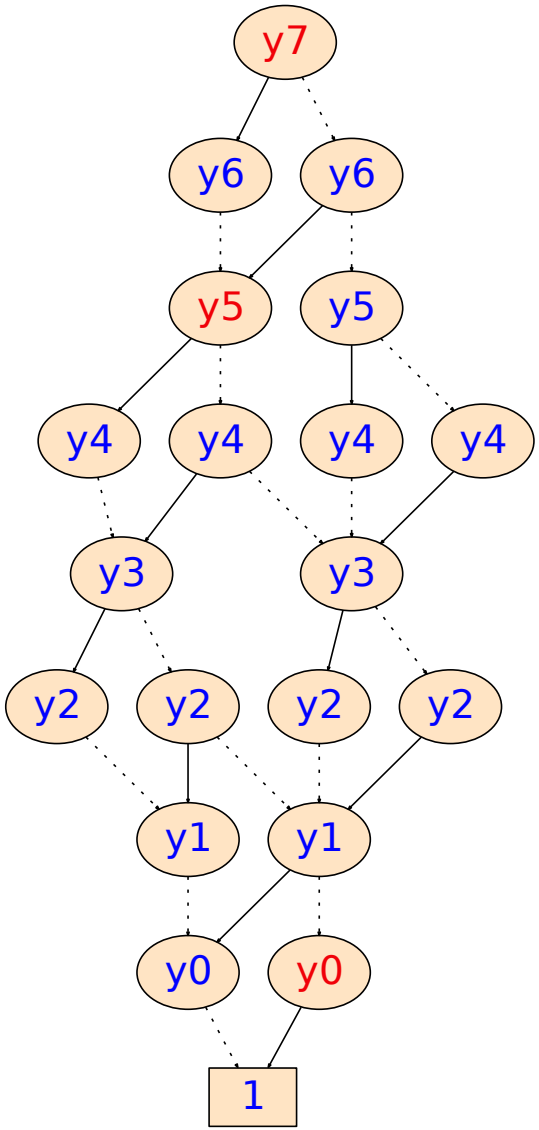
Example



Permutation	0	1	2	3	4	5	6	7
PS := PerfectShuffle	0	2	4	6	1	3	5	7
FS := FlipShuffle	0	4	1	5	2	6	3	7
BF := Butterfly	0	4	2	6	1	5	3	7
MR := Mirror	7	6	5	4	3	2	1	0
RM := RevMirror	7	3	5	1	6	2	4	0
OP := OutputPerm	7	0	5	2	3	4	1	6

- here, we consider input vectors with 2 valid bits
- they must be routed to output $z = (0, ..., 0, 1, 1)$ for sorting
- the BDD tells us that permutations must map e.g. y_0, y_7 to z_7, z_6

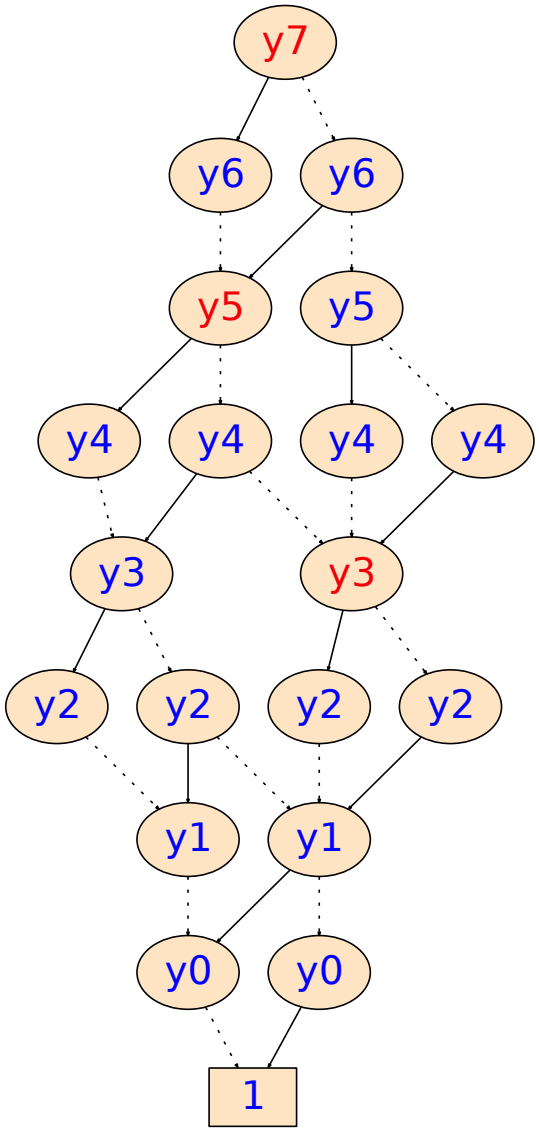
Example



Permutation	0	1	2	3	4	5	6	7
PS := PerfectShuffle	0	2	4	6	1	3	5	7
FS := FlipShuffle	0	4	1	5	2	6	3	7
BF := Butterfly	0	4	2	6	1	5	3	7
MR := Mirror	7	6	5	4	3	2	1	0
RM := RevMirror	7	3	5	1	6	2	4	0
OP := OutputPerm	7	0	5	2	3	4	1	6

- here, we consider input vectors with 3 valid bits
- they must be routed to output $z = (0, ..., 0, 1, 1, 1)$ for sorting
- the BDD tells us that permutations must map e.g. y_0, y_7, y_5 to z_7, z_6, z_5
- none of the above permutations can do that
- backtrack and find another permutation

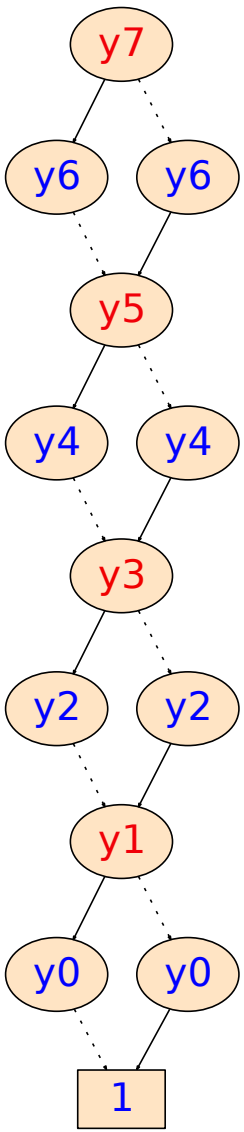
Example



Permutation	0	1	2	3	4	5	6	7
PS := PerfectShuffle	0	2	4	6	1	3	5	7
FS := FlipShuffle	0	4	1	5	2	6	3	7
BF := Butterfly	0	4	2	6	1	5	3	7
MR := Mirror	7	6	5	4	3	2	1	0
RM := RevMirror	7	3	5	1	6	2	4	0
OP := OutputPerm	7	0	5	2	3	4	1	6

- here, we consider again input vectors with 3 valid bits
- they must be routed to output $z = (0, ..., 0, 1, 1, 1)$ for sorting
- the BDD tells us that permutations must map e.g. y_7, y_3, y_5 to z_7, z_6, z_5 (like BF)

Example



Permutation	0	1	2	3	4	5	6	7
PS := PerfectShuffle	0	2	4	6	1	3	5	7
FS := FlipShuffle	0	4	1	5	2	6	3	7
BF := Butterfly	0	4	2	6	1	5	3	7
MR := Mirror	7	6	5	4	3	2	1	0
RM := RevMirror	7	3	5	1	6	2	4	0
OP := OutputPerm	7	0	5	2	3	4	1	6

- finally, we extend it to input vectors with 4 valid bits
- they must be routed to output $z = (0, ..., 0, 1, 1, 1, 1)$ for sorting
- the BDD tells us that permutations must map e.g. y_7, y_3, y_5, y_1 to z_7, z_6, z_5, z_4 (like BF)

Experimental Results

	MS	RB	BY
PS	✓($S1$)	–	✓($S1$)
FS	✓($S2$)	✓($S1$)	–
BF	–	✓($S1$)	✓($S2$)
MR	–	–	–
RM	–	✓($S1$)	✓($S2$)
OP	–	✓($S1$)	✓($S3$)

- we found three BDD sequences $S1, S2, S3$
- these correspond with three classes of potential output permutations for sorting and concentrating
- we made also a full analysis over all 8! permutations for 8 inputs

Outline

Conclusions and Future Work

conclusions:

- Narasimha showed that RB-FS-BF can be configured as sorter and as concentrator
- with our method, one can verify which other networks can alternatively be used

future work:

- determine the control logic of the switches
- determine minimal number of required switches

The End

Thank you for your attention.

Questions?