

Decisions, Decisions:

**Using Decision Diagrams to Encode and Manipulate
Large Structured Data**

Gianfranco Ciardo

University of California, Riverside

Question:

How would you store a vector of size 10^{100} ?

Before answering, you might want to ask:

Where does it come from?

If it comes from sampling, from digitizing a picture, etc.:

Use compression techniques

But what if it comes from some kind of man-made structured artifact or abstraction?

The possible ways to organize N objects into M bins

A minimal set of linearly-independent solutions $\mathbf{x} \in \mathbb{N}^m$ to $\mathbf{A} \cdot \mathbf{x} = \mathbf{0}$, where $\mathbf{A} \in \mathbb{N}^{n \times m}$

The set of reachable states during the execution of a distributed program

These problems are described in a relatively compact way, but their answer is combinatorially large

Decision diagrams for boolean functions

A structured discrete-state model is specified by

- a **potential state space** $\mathcal{X}_{pot} = \mathcal{X}_L \times \cdots \times \mathcal{X}_1$
 - the “type” of the (global) state
 - \mathcal{X}_k is the (discrete) **local state space** for the k^{th} submodel
 - if \mathcal{X}_k is finite, we can map it to $\{0, 1, \dots, n_k - 1\}$ n_k might be unknown a priori
- a set of **initial states** $\mathcal{X}_{init} \subseteq \mathcal{X}_{pot}$
 - often there is a single initial state \mathbf{i}_{init}
- a set of **events** \mathcal{E} defining a **disjunctively-partitioned next-state function** or transition relation
 - $\mathcal{N}_\alpha : \mathcal{X}_{pot} \rightarrow 2^{\mathcal{X}_{pot}}$ $\mathbf{j} \in \mathcal{N}_\alpha(\mathbf{i})$ iff state \mathbf{j} can be reached by **firing** event α in state \mathbf{i}
 - $\mathcal{N} : \mathcal{X}_{pot} \rightarrow 2^{\mathcal{X}_{pot}}$ $\mathcal{N}(\mathbf{i}) = \bigcup_{\alpha \in \mathcal{E}} \mathcal{N}_\alpha(\mathbf{i})$
 - naturally extended to sets of states $\mathcal{N}_\alpha(\mathcal{Y}) = \bigcup_{\mathbf{i} \in \mathcal{Y}} \mathcal{N}_\alpha(\mathbf{i})$ and $\mathcal{N}(\mathcal{Y}) = \bigcup_{\mathbf{i} \in \mathcal{Y}} \mathcal{N}(\mathbf{i})$
 - α is **enabled** in \mathbf{i} iff $\mathcal{N}_\alpha(\mathbf{i}) \neq \emptyset$, otherwise it is **disabled**
 - \mathbf{i} is **absorbing**, or **dead**, if $\mathcal{N}(\mathbf{i}) = \emptyset$

The **state space** \mathcal{X}_{reach} of the model is the smallest subset of \mathcal{X}_{pot} containing \mathcal{X}_{init} and satisfying:

- the **recursive definition** $\mathbf{i} \in \mathcal{X}_{reach} \wedge \mathbf{j} \in \mathcal{N}(\mathbf{i}) \Rightarrow \mathbf{j} \in \mathcal{X}_{reach}$
- or the **fixed-point equation** $\mathcal{Y} = \mathcal{Y} \cup \mathcal{N}(\mathcal{Y})$

$$\mathcal{X}_{reach} = \mathcal{X}_{init} \cup \mathcal{N}(\mathcal{X}_{init}) \cup \mathcal{N}^2(\mathcal{X}_{init}) \cup \mathcal{N}^3(\mathcal{X}_{init}) \cup \dots = \mathcal{N}^*(\mathcal{X}_{init})$$

```
ExploreExplicit(  $\mathcal{X}_{init}$  : set of states,  $\mathcal{N}$  : next-state function ) : set of states
local  $\mathcal{U}, \mathcal{Y}$  : set of states;
local  $\mathbf{i}, \mathbf{j}$  : state;
1  $\mathcal{Y} \leftarrow \emptyset$ ;                                 $\mathcal{Y}$  contains the known states already explored
2  $\mathcal{U} \leftarrow \mathcal{X}_{init}$ ;                           $\mathcal{U}$  contains the known states not yet explored
3 while  $\mathcal{U} \neq \emptyset$  do
4     choose a state  $\mathbf{i}$  in  $\mathcal{U}$  and move it from  $\mathcal{U}$  to  $\mathcal{Y}$ ;
5     for each  $\mathbf{j} \in \mathcal{N}(\mathbf{i})$  do
6         if  $\mathbf{j} \notin \mathcal{Y} \cup \mathcal{U}$  then                search to determine whether  $\mathbf{j}$  is a new state
7              $\mathcal{U} \leftarrow \mathcal{U} \cup \{\mathbf{j}\}$ ;        remember to explore  $\mathbf{j}$  later
8         end if;
9     end for;
10 end while;
11 return  $\mathcal{Y}$ ;
```

the memory requirements are $O(|\mathcal{X}_{reach}|)$

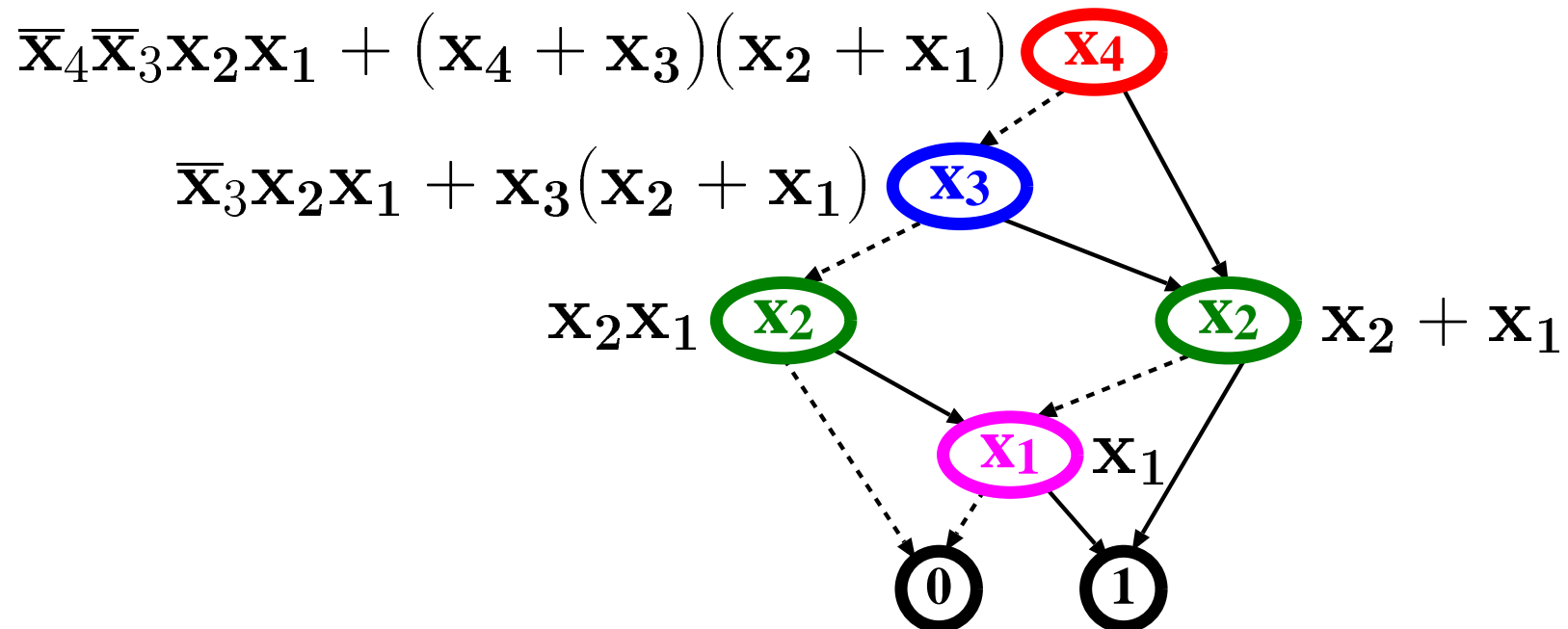
most time is spent searching for a state (line 6)

“Graph-based algorithms for boolean function manipulation”

Randy Bryant (Carnegie Mellon University)

IEEE Transactions on Computers, 1986

CiteSeer most cited document!



BDDs are a **canonical** representation of boolean functions $f : \mathbb{B}^L \rightarrow \mathbb{B}$

Assume a **domain** $\mathcal{X}_{pot} = \mathcal{X}_L \times \cdots \times \mathcal{X}_1$, where $\mathcal{X}_k = \{0, 1, \dots, n_k - 1\}$, for some $n_k \in \mathbb{N}$

Assume the **range** $\mathcal{X}_0 = \mathbb{B}$

An MDD is an acyclic directed edge-labeled graph where:

- The only **terminal** nodes can be **0** and **1**, and are at **level 0** $\mathbf{0.lvl} = \mathbf{1.lvl} = 0$
- A **nonterminal** node p is at a **level** k , with $L \geq k \geq 1$ $p.lvl = k$
- For each $i_k \in \mathcal{X}_k$, a nonterminal node p at level k has an outgoing edge pointing to **child** $p[i_k]$
- The level of a child is lower than that of p $p[i_k].lvl < p.lvl$
- A node p at level k encodes the **function** $v_p : \mathcal{X}_{pot} \rightarrow \mathbb{B}$ defined recursively by

$$v_p(x_L, \dots, x_1) = \begin{cases} p & \text{if } k = 0 \\ v_{p[x_k]}(x_L, \dots, x_1) & \text{if } k > 0 \end{cases}$$

Instead of levels, we can also talk of **variables**:

- The terminal nodes are associated with the **range variable** x_0
- A nonterminal node is associated with a **domain variable** x_k , with $L \geq k \geq 1$

For **canonical** MDDs, we further require that

- There are no **duplicates**: if $p.lvl = q.lvl = k$ and $p[i_k] = q[i_k]$ for all $i_k \in \mathcal{X}_k$, then $p = q$

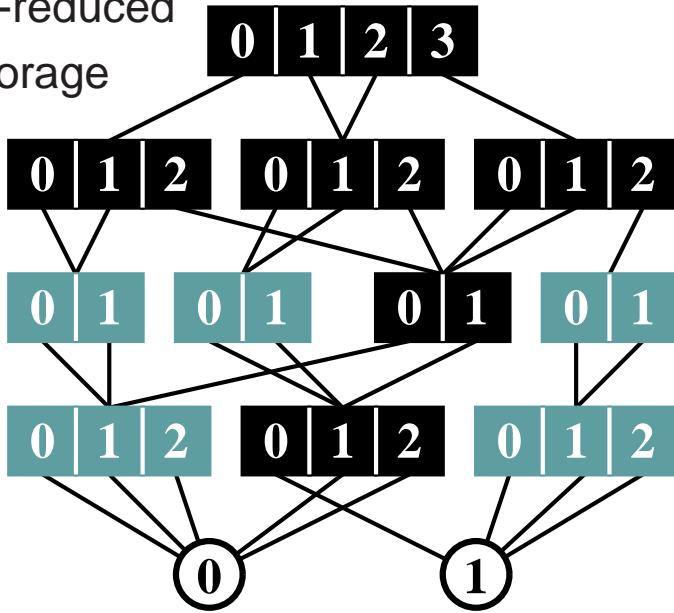
Then, if the MDD is **quasi-reduced**, there is **no level skipping**:

- The only **root** nodes with no incoming arcs are at level L
- If a node p is at level k , each child $p[i_k]$ is at level $k - 1$

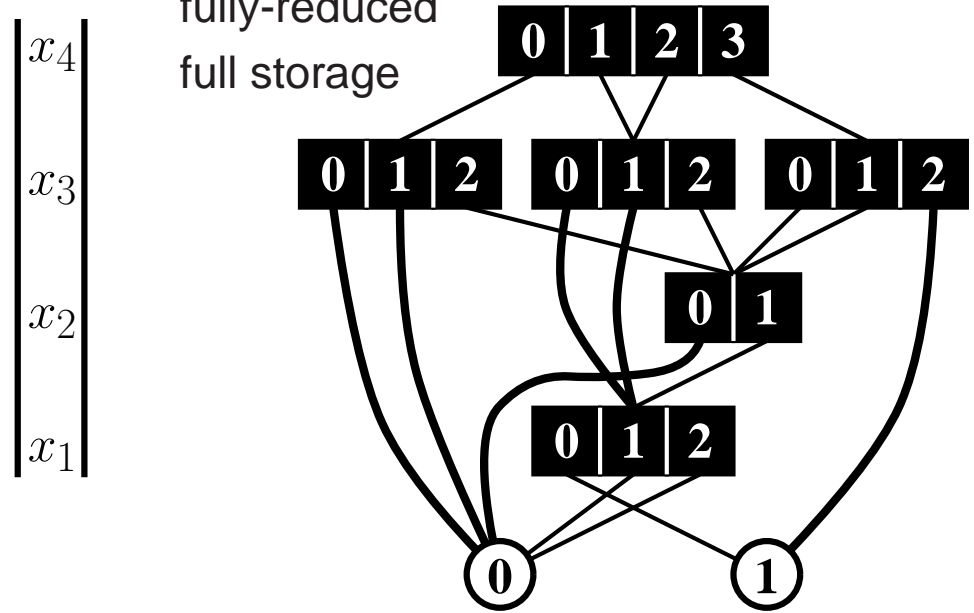
Or, if the MDD is **fully-reduced**, there is **maximum level skipping**:

- There are no **redundant** nodes p at level k satisfying $p[i_k] = q$ for all $i_k \in \mathcal{X}_k$

quasi-reduced
full storage



fully-reduced
full storage



What if we don't know the range of each \mathcal{X}_k ?

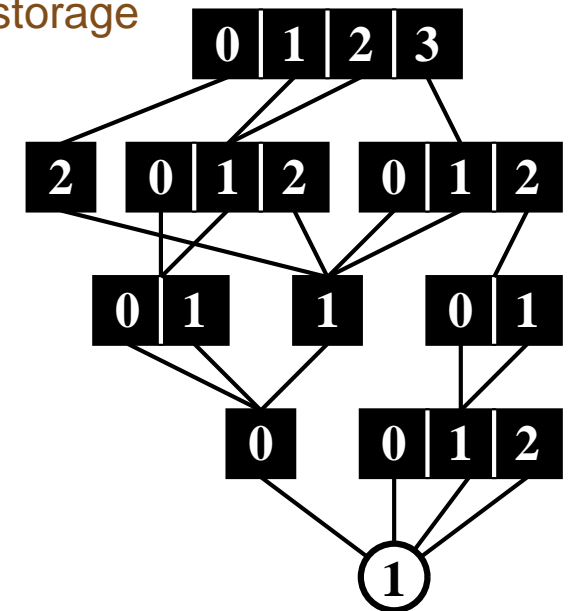
We can simply assume $\mathcal{X}_{pot} = \mathbb{N}^L$

All but a finite number of edges point to **0**

Only nodes encoding \emptyset are redundant

x_4
 x_3
 x_2
 x_1

sparse storage



Fully-reduced MDDs: each node in the MDD encodes a different function

Quasi-reduced MDDs: each node at a given level of the MDD encodes a different function

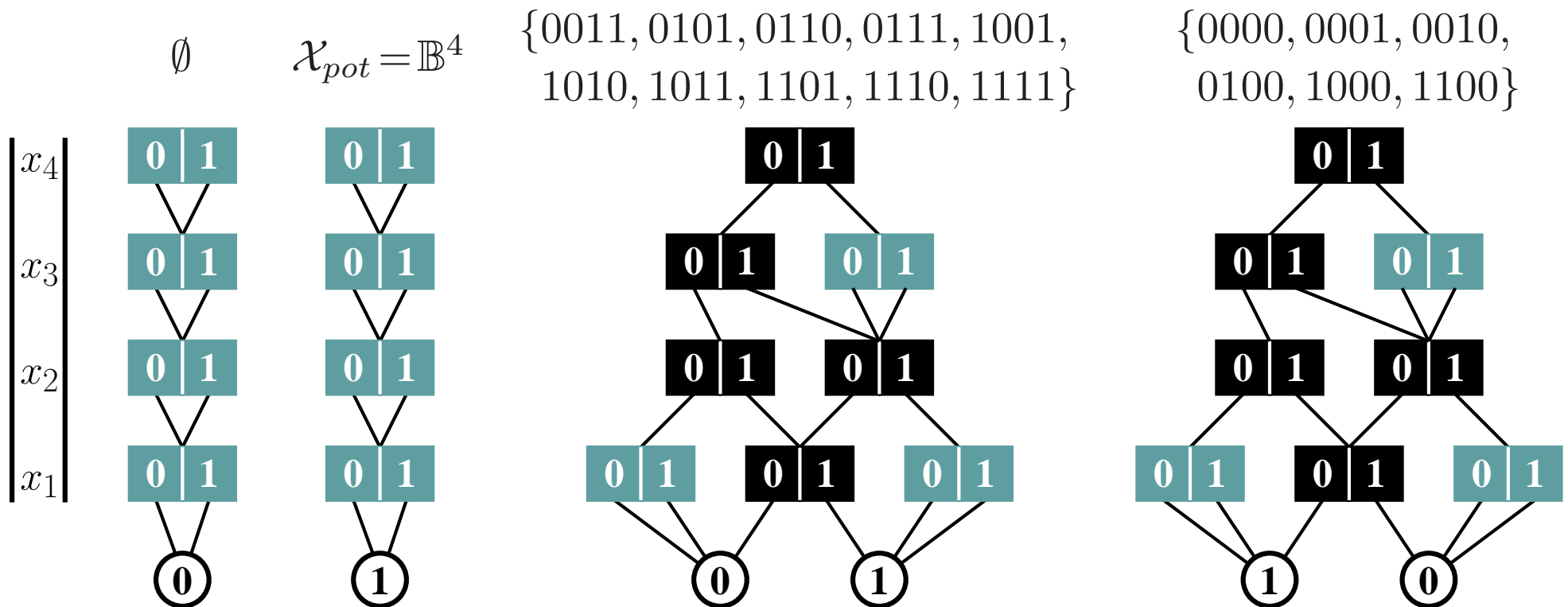
- Given a function $f : \mathcal{X}_{pot} \rightarrow \mathbb{B}$, there is a unique MDD encoding it (for an order x_L, \dots, x_1)
- Many functions have a **very compact** MDD encoding
- The constant functions 0 and 1 are represented by the nodes **0** and **1**, respectively
- Given the MDD encoding function f : test whether $f \equiv 0$ or $f \equiv 1$ in $O(1)$ time
- Given the MDDs encoding functions f and g : test whether $f \equiv g$ in $O(1)$ time

- The variable ordering affects the size of the MDD, consider $x_L \Leftrightarrow y_L \wedge \dots \wedge x_1 \Leftrightarrow y_1$
 - with the order $(x_L, y_L, \dots, x_1, y_1)$ $O(L)$ nodes
 - with the order $(x_L, \dots, x_1, y_L, \dots, y_1)$ $O(2^L)$ nodes
- The MDD encoding of some functions is exponentially large for any order
 - the expression for bit 32 of the 64-bit result of the multiplication of two 32-bit integers
- Finding the optimal ordering that minimizes the MDD size is an NP-complete problem

We can encode a set $\mathcal{Y} \subseteq \mathcal{X}_{pot}$ as an MDD p through its **characteristic function**:

$$\mathbf{i} = (i_L, \dots, i_1) \in \mathcal{Y} \Leftrightarrow v_p(i_L, \dots, i_1) = 1$$

The **size of the set** encoded by MDD p is not directly related to the **size of the MDD** itself



mdd Union(lvl k , *mdd* p , *mdd* q) is

```

local mdd  $r, r_0, \dots, r_{n_k-1}$ ;
1 if  $p = \mathbf{0}$  then return  $q$ ;
2 if  $q = \mathbf{0}$  then return  $p$ ;
3 if  $p = q$  then return  $p$ ;
4 if Cache contains entry  $\langle \text{UnionCODE}, \{p, q\} : r \rangle$  then return  $r$ ;
5 foreach  $i_k \in \mathcal{X}_k$  do
6    $r_{i_k} \leftarrow \text{Union}(k-1, p[i_k], q[i_k])$ ;
7 end for
8  $r \leftarrow \text{UniqueTableInsert}(k, r_0, \dots, r_{n_k-1})$ ;
9 enter  $\langle \text{UnionCODE}, \{p, q\} : r \rangle$  in Cache;
10 return  $r$ ;
    
```

Intersection(k, p, q) differs from *Union*(k, p, q) only in the terminal cases:

Union: if $p = \mathbf{0}$ then return q ;
 if $q = \mathbf{0}$ then return p ;

Intersection: if $p = \mathbf{1}$ then return q ;
 if $q = \mathbf{1}$ then return p ;

complexity:

$$O\left(\sum_{L \geq k \geq 1} \#(\text{nodes at level } k \text{ in } p) \times \#(\text{nodes at level } k \text{ in } q)\right)$$

Given an L -level MDD on (x_L, \dots, x_1) rooted at p_* , encoding a set $\mathcal{Y} \subseteq \mathcal{X}_{pot}$

Given a $2L$ -level MDD on $(x_L, x'_L, \dots, x_1, x'_1)$ rooted at r_* , encoding a function $\mathcal{N} : \mathcal{X}_{pot} \rightarrow 2^{\mathcal{X}_{pot}}$

RelationalProduct (L, p_*, r_*) returns the root of the MDD encoding $\{\mathbf{j} : \exists \mathbf{i} \in \mathcal{Y} \wedge \mathbf{j} \in \mathcal{N}(\mathbf{i})\}$

mdd RelationalProduct(lvl k , mdd p , mdd2 r) is

quasi-reduced version

local mdd q, q_0, \dots, q_{n_k-1} ;

1 if $k = 0$ then return $p \wedge r$;

2 if *Cache* contains entry $\langle \text{RelationalProductCODE}, p, r : q \rangle$ then return q ;

3 foreach $j_k \in \mathcal{X}_k$ do

4 $q_{j_k} \leftarrow \mathbf{0}$;

5 foreach i_k such that $r[i_k][j_k] \neq \mathbf{0}$ do

6 $q_{j_k} \leftarrow \text{Union}(q_{j_k}, \text{RelationalProduct}(k-1, p[i_k], r[i_k][j_k]))$;

7 $q \leftarrow \text{UniqueTableInsert}(k, q_0, \dots, q_{n_k-1})$;

8 enter $\langle \text{RelationalProductCODE}, p, r : q \rangle$ in *Cache*;

9 return q ;

An L -level MDD encodes a set of states $\mathcal{Y} \subseteq \mathcal{X}_{pot} = \mathcal{X}_L \times \dots \times \mathcal{X}_1$

$\mathbf{i} \equiv (i_L, \dots, i_1) \in \mathcal{Y} \Leftrightarrow$ the path from the root corresponding to \mathbf{i} leads to terminal $\mathbf{1}$

A $2L$ -level MDD encodes the next-state function $\mathcal{N} : \mathcal{X}_{pot} \rightarrow 2^{\mathcal{X}_{pot}}$

$\mathbf{j} \in \mathcal{N}(\mathbf{i}) \Leftrightarrow$ the path from the root corresponding to the interleaving of \mathbf{i} and \mathbf{j} leads to terminal $\mathbf{1}$

The state space \mathcal{X}_{reach} is the fixpoint of the iteration

$$\mathcal{X}_{init} \cup \mathcal{N}(\mathcal{X}_{init}) \cup \mathcal{N}(\mathcal{N}(\mathcal{X}_{init})) \cup \mathcal{N}(\mathcal{N}(\mathcal{N}(\mathcal{X}_{init}))) \cup \dots$$

Standard method

Alternative *All* method

ExploreMdd($\mathcal{X}_{init}, \mathcal{N}$) is

```

1  $\mathcal{Y} \leftarrow \mathcal{X}_{init};$            known states
2  $\mathcal{U} \leftarrow \mathcal{X}_{init};$        unexplored states
3 repeat
4    $\mathcal{W} \leftarrow \mathcal{N}(\mathcal{U});$      potentially new states
5    $\mathcal{U} \leftarrow \mathcal{W} \setminus \mathcal{Y};$    truly new states
6    $\mathcal{Y} \leftarrow \mathcal{Y} \cup \mathcal{U};$ 
7 until  $\mathcal{U} = \emptyset;$ 
8 return  $\mathcal{Y};$ 
    
```

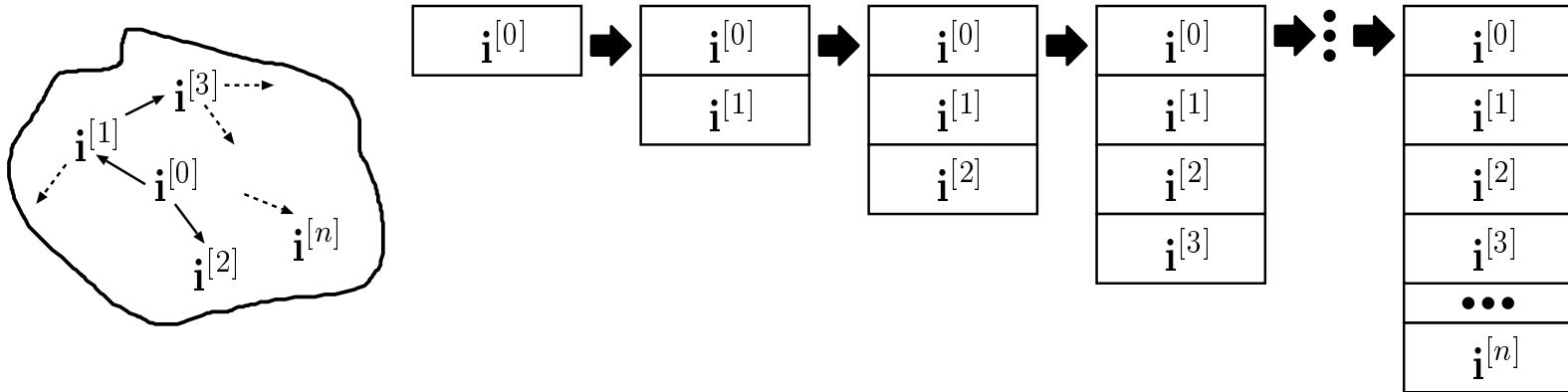
AllExploreMdd($\mathcal{X}_{init}, \mathcal{N}$) is

```

1  $\mathcal{Y} \leftarrow \mathcal{X}_{init};$ 
2 repeat
3    $\mathcal{O} \leftarrow \mathcal{Y};$            old states
4    $\mathcal{Y} \leftarrow \mathcal{O} \cup \mathcal{N}(\mathcal{O});$  new states
5 until  $\mathcal{O} = \mathcal{Y};$ 
6 return  $\mathcal{Y};$ 
    
```

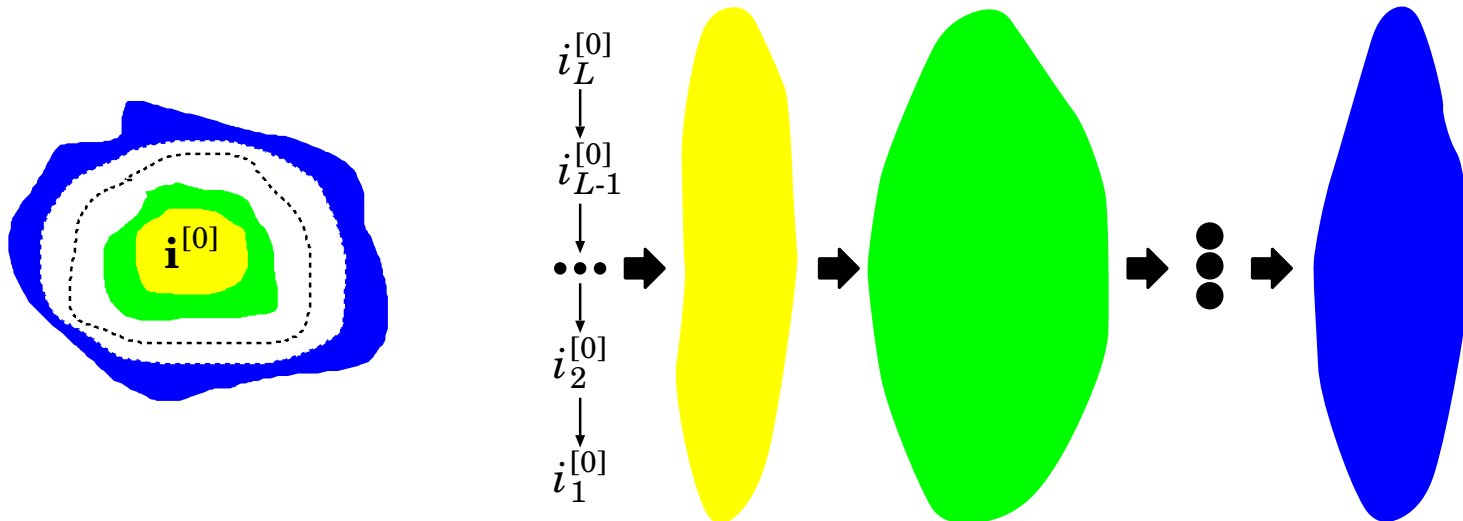
Explicit generation of the state space \mathcal{X}_{reach} adds **one state** at a time

- memory $O(\text{states})$, increases linearly, peaks at the end



Symbolic generation of the state space \mathcal{X}_{reach} with decision diagrams adds **sets of states** instead

- memory $O(\text{decision diagram nodes})$, grows and shrinks, usually peaks well before the end



If \mathcal{N} is stored in a disjunctively partitioned form as $\mathcal{N} = \bigcup_{\alpha \in \mathcal{E}} \mathcal{N}_\alpha$, using $|\mathcal{E}|$ MDDs, the effect of

$\mathcal{W} \leftarrow \mathcal{N}(\mathcal{U});$ *potentially new states*

$\mathcal{U} \leftarrow \mathcal{W} \setminus \mathcal{Y};$ *truly new states*

is exactly achieved with the statements

$\mathcal{W} \leftarrow \emptyset;$

for each $\alpha \in \mathcal{E}$ do

$\mathcal{W} \leftarrow \mathcal{W} \cup \mathcal{N}_\alpha(\mathcal{U});$

$\mathcal{U} \leftarrow \mathcal{W} \setminus \mathcal{Y};$

However, if we do not require strict breadth-first order, we can use **chaining** and do

for each $\alpha \in \mathcal{E}$ do

$\mathcal{U} \leftarrow \mathcal{U} \cup \mathcal{N}_\alpha(\mathcal{U});$

$\mathcal{U} \leftarrow \mathcal{U} \setminus \mathcal{Y};$

BfSsGen($\mathcal{X}_{init}, \{\mathcal{N}_\alpha : \alpha \in \mathcal{E}\}$)

```

1  $\mathcal{Y} \leftarrow \mathcal{X}_{init};$ 
2  $\mathcal{U} \leftarrow \mathcal{X}_{init};$ 
3 repeat
4    $\mathcal{W} \leftarrow \emptyset;$ 
5   for each  $\alpha \in \mathcal{E}$  do
6      $\mathcal{W} \leftarrow \mathcal{W} \cup \mathcal{N}_\alpha(\mathcal{U});$ 
7    $\mathcal{U} \leftarrow \mathcal{W} \setminus \mathcal{Y};$ 
8    $\mathcal{Y} \leftarrow \mathcal{Y} \cup \mathcal{U};$ 
9 until  $\mathcal{U} = \emptyset;$ 
10 return  $\mathcal{Y};$ 

```

ChSsGen($\mathcal{X}_{init}, \{\mathcal{N}_\alpha : \alpha \in \mathcal{E}\}$)

```

1  $\mathcal{Y} \leftarrow \mathcal{X}_{init};$ 
2  $\mathcal{U} \leftarrow \mathcal{X}_{init};$ 
3 repeat
4   for each  $\alpha \in \mathcal{E}$  do
5      $\mathcal{U} \leftarrow \mathcal{U} \cup \mathcal{N}_\alpha(\mathcal{U});$ 
6      $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathcal{Y};$ 
7    $\mathcal{Y} \leftarrow \mathcal{Y} \cup \mathcal{U};$ 
8 until  $\mathcal{U} = \emptyset;$ 
9 return  $\mathcal{Y};$ 

```

AllBfSsGen($\mathcal{X}_{init}, \{\mathcal{N}_\alpha : \alpha \in \mathcal{E}\}$)

```

1  $\mathcal{Y} \leftarrow \mathcal{X}_{init};$ 
2 repeat
3    $\mathcal{O} \leftarrow \mathcal{Y};$ 
4    $\mathcal{W} \leftarrow \emptyset;$ 
5   for each  $\alpha \in \mathcal{E}$  do
6      $\mathcal{W} \leftarrow \mathcal{W} \cup \mathcal{N}_\alpha(\mathcal{O});$ 
7    $\mathcal{Y} \leftarrow \mathcal{O} \cup \mathcal{W};$ 
8 until  $\mathcal{O} = \mathcal{Y};$ 
9 return  $\mathcal{Y};$ 

```

AllChSsGen($\mathcal{X}_{init}, \{\mathcal{N}_\alpha : \alpha \in \mathcal{E}\}$)

```

1  $\mathcal{Y} \leftarrow \mathcal{X}_{init};$ 
2 repeat
3    $\mathcal{O} \leftarrow \mathcal{Y};$ 
4   for each  $\alpha \in \mathcal{E}$  do
5      $\mathcal{Y} \leftarrow \mathcal{Y} \cup \mathcal{N}_\alpha(\mathcal{Y});$ 
6 until  $\mathcal{O} = \mathcal{Y};$ 
7 return  $\mathcal{Y};$ 

```

N	$ \mathcal{X}_{reach} $	Time (sec)				Memory (MB)				final
		Bf	$AllBf$	Ch	$AllCh$	Bf	$AllBf$	Ch	$AllCh$	

Dining Philosophers: $L = N/2$, $|\mathcal{X}_k| = 34$ for all k

50	2.2×10^{31}	37.6	36.8	1.3	1.3	146.8	131.6	2.2	2.2	<0.1
100	5.0×10^{62}	644.1	630.4	5.4	5.3	>999.9	>999.9	8.9	8.9	<0.1
1000	9.2×10^{626}	—	—	895.4	915.5	—	—	895.2	895.0	0.3

Slotted Ring Network: $L = N$, $|\mathcal{X}_k| = 15$ for all k

5	5.3×10^4	0.2	0.3	0.1	0.1	0.8	1.1	0.3	0.2	<0.1
10	8.3×10^9	21.5	24.1	2.1	1.2	39.0	45.0	5.7	3.3	<0.1
15	1.5×10^{15}	745.4	771.5	18.5	8.9	344.3	375.4	35.1	20.2	<0.1

Round Robin Mutual Exclusion: $L = N + 1$, $|\mathcal{X}_k| = 10$ for all k except $|\mathcal{X}_1| = N + 1$

10	2.3×10^4	0.2	0.3	0.1	0.1	0.6	1.2	0.1	0.1	<0.1
20	4.7×10^7	2.7	4.4	0.3	0.3	5.9	12.8	0.5	0.5	<0.1
50	1.3×10^{17}	263.2	427.6	2.9	2.8	126.7	257.7	4.3	3.8	0.1

FMS: $L = 19$, $|\mathcal{X}_k| = N + 1$ for all k except $|\mathcal{X}_{17}| = 4$, $|\mathcal{X}_{12}| = 3$, $|\mathcal{X}_7| = 2$

5	2.9×10^6	0.7	0.7	0.1	0.1	2.6	2.2	0.4	0.2	<0.1
10	2.5×10^9	7.0	5.8	0.5	0.3	18.2	14.7	2.3	1.3	<0.1
25	8.5×10^{13}	677.2	437.9	12.9	5.1	319.7	245.3	42.7	21.2	0.1

Given an event $\alpha \in \mathcal{E}$, consider the subset of the state variables $\{x_L, \dots, x_1\}$ that:

- can be modified by α : $\mathcal{V}_M(t) = \{x_k : \exists \mathbf{i}, \mathbf{i}' \in \mathcal{X}_{pot}, \mathbf{i}' \in \mathcal{N}_t(\mathbf{i}) \wedge \mathbf{i}[k] \neq \mathbf{i}'[k]\}$
- can disable α : $\mathcal{V}_D(t) = \{x_k : \exists \mathbf{i}, \mathbf{j} \in \mathcal{X}_{pot}, \forall h \neq k, \mathbf{i}[h] = \mathbf{j}[h] \wedge \mathcal{N}_t(\mathbf{i}) \neq \emptyset \wedge \mathcal{N}_t(\mathbf{j}) = \emptyset\}$

If $x_k \notin \mathcal{V}_M \cup \mathcal{V}_D$, we say that event α and variable x_k , or level k , are independent

Most events in a globally-asynchronous locally-synchronous model are highly localized:

- Let $Top(\alpha) = \max\{k : x_k \in \mathcal{V}_M(\alpha) \cup \mathcal{V}_D(\alpha)\}$ be the highest level dependent on α
- Let $Bot(\alpha) = \min\{k : x_k \in \mathcal{V}_M(\alpha) \cup \mathcal{V}_D(\alpha)\}$ be the lowest level dependent on α
- The span (of levels) $\{Top(\alpha), \dots, Bot(\alpha)\}$ for event α is often much smaller than $\{L, \dots, 1\}$

fully-reduced $2L$ -level MDD encoding of \mathcal{N} does not exploit locality

need **Kronecker**, **identity-reduced** $2L$ -level MDD, or **MxD** encoding

An MDD node p at level k is **saturated** if it encodes a fixed point w.r.t. any α such that $Top(\alpha) \leq k$ (this implies that all nodes reachable from p are also saturated)

- build the L -level MDD encoding of \mathcal{X}_{init} if $|\mathcal{X}_{init}| = 1$, there is one node per level
- saturate each node at level 1: fire in them all events α such that $Top(\alpha) = 1$
- saturate each node at level 2: fire in them all events α such that $Top(\alpha) = 2$
(if this creates nodes at level 1, saturate them immediately upon creation)
- saturate each node at level 3: fire in them all events α such that $Top(\alpha) = 3$
(if this creates nodes at levels 2 or 1, saturate them immediately upon creation)
- ...
- saturate the root node at level L : fire in it all events α such that $Top(\alpha) = L$
(if this creates nodes at levels $L-1, L-2, \dots, 1$, saturate them immediately upon creation)

states are **not** discovered in breadth-first order

enormous time and memory savings for asynchronous systems

mdd $Saturate(\text{level } k, \text{mdd } p)$ is

quasi-reduced version

local mdd $r, r_0, \dots, r_{n_k-1};$

1 if $k = 0$ then return p ;

2 if $Cache$ contains entry $\langle SaturateCODE, p : r \rangle$ then return r ;

3 foreach $i_k \in \mathcal{X}_k$ do

4 $r_{i_k} \leftarrow Saturate(k-1, p[i_k]);$

first, be sure that the children are saturated

5 repeat

6 choose $\alpha \in \mathcal{E}, i_k, j_k \in \mathcal{X}_k$ s.t. $Top(\alpha) = k$ and $r_{i_k} \neq \mathbf{0}$ and $\mathcal{N}_\alpha[i_k][j_k] \neq \mathbf{0}$;

7 $r_{j_k} \leftarrow Union(r_{j_k}, RelProdSat(k-1, r_{i_k}, \mathcal{N}_\alpha[i_k][j_k]));$

8 until r_0, \dots, r_{n_k-1} do not change;

9 $r \leftarrow UniqueTableInsert(k, r_0, \dots, r_{n_k-1});$

10 enter $\langle SaturateCODE, p : r \rangle$ in $Cache$;

11 return r ;

mdd $RelProdSat(\text{level } k, \text{mdd } q, \text{mdd2 } f)$ is

local mdd $r, r_0, \dots, r_{n_k-1};$

1 if $k = 0$ then return $q \wedge f$;

2 if $Cache$ contains entry $\langle RelProdSatCODE, q, f : r \rangle$ then return r ;

3 foreach $i_k, j_k \in \mathcal{X}_k$ s.t. $q[i_k] \neq \mathbf{0}$ and $f[i_k][j_k] \neq \mathbf{0}$ do

4 $r_{j_k} \leftarrow Union(r_{j_k}, RelProdSat(k-1, q[i_k], f[i_k][j_k]));$

5 $r \leftarrow Saturate(k, UniqueTableInsert(k, r_0, \dots, r_{n_k-1}));$

6 enter $\langle RelProdSatCODE, q, f : r \rangle$ in $Cache$;

7 return r .

N	$ \mathcal{X}_{reach} $	Peak memory (kB)		Time (sec)	
		SMART	NuSMV	SMART	NuSMV

Dining Philosophers: $L = N$

50	2.23×10^{31}	22	10,819	0.15	5.9
200	2.47×10^{125}	93	72,199	0.68	12,905.7
10,000	4.26×10^{6269}	4,686	—	877.82	—

Slotted Ring Network: $L = N$

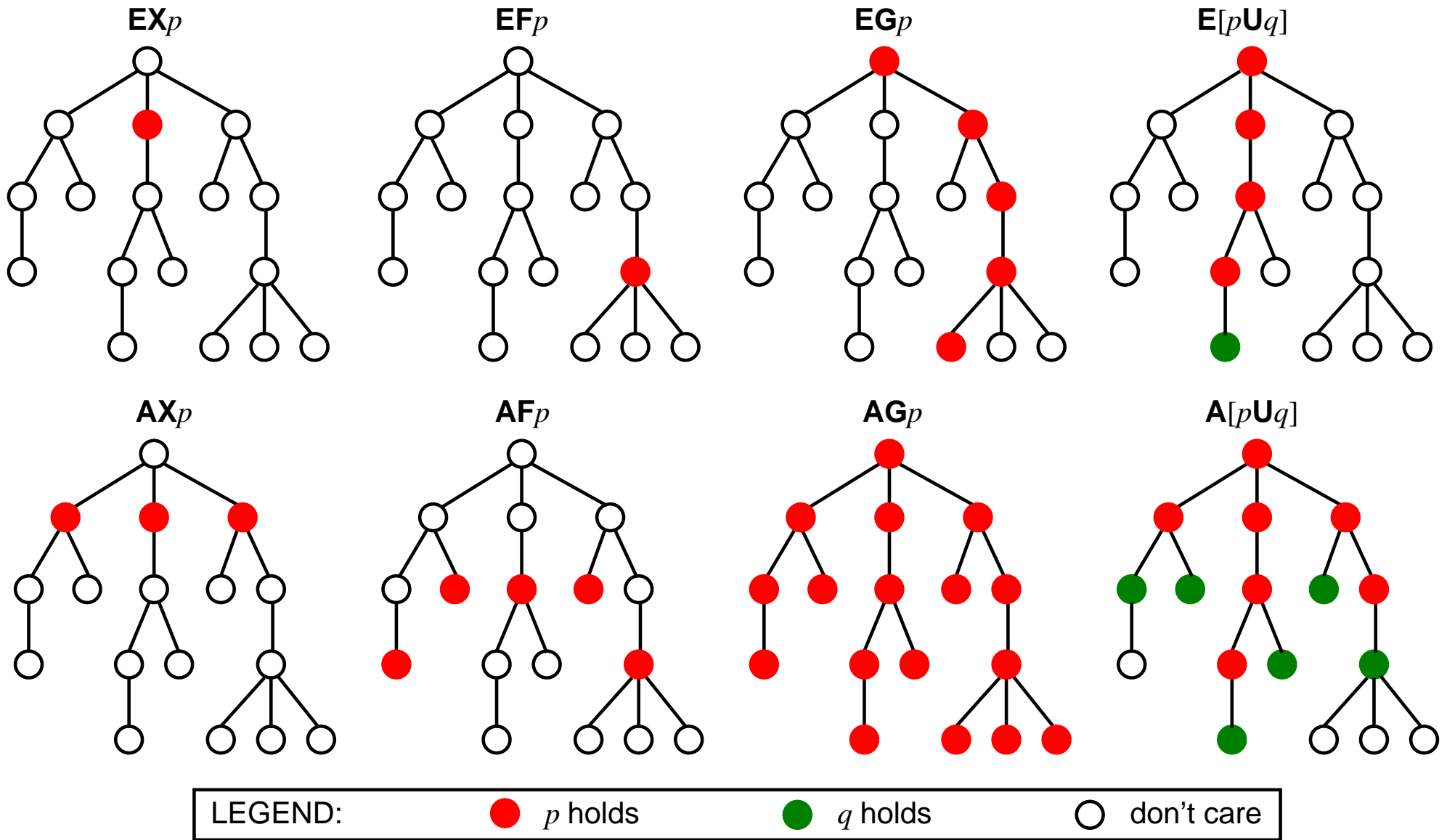
10	8.29×10^9	28	10,819	0.13	5.5
15	1.46×10^{15}	80	13,573	0.39	2,039.5
200	8.38×10^{211}	120,316	—	902.11	—

Round Robin Mutual Exclusion: $L = N + 1$

20	4.72×10^7	20	7,306	0.07	0.8
100	2.85×10^{32}	372	26,628	3.81	2,475.3
300	1.37×10^{93}	3,109	—	140.98	—

Flexible Manufacturing System: $L = 19$

10	4.28×10^6	26	11,238	0.05	9.4
20	3.84×10^9	101	31,718	0.20	1,747.8
250	3.47×10^{26}	69,087	—	231.17	—



EX, EU, and EG form a **complete set** of CTL operators, since:

- $AXp = \neg EX\neg p$
- $EFp = E[true \text{ U } p]$
- $AGp = \neg EF\neg p$
- $E[pRq] = \neg A[\neg p \text{ U } \neg q]$
- $AFp = \neg EG\neg p$
- $A[p \text{ U } q] = \neg E[\neg q \text{ U } \neg p \wedge \neg q] \wedge \neg EG\neg q$
- $A[pRq] = \neg E[\neg p \text{ U } \neg q]$

The system always reaches and remains in a stable state after an initial startup period

- $initial \Rightarrow AF (A \text{ stable } U \text{ shutdown})$

At any point in the execution, it must be possible to return to a reset state

- $EF \text{ reset}$

If a process ask access to the critical region, it eventually obtains it

- $request_critical \Rightarrow AF \text{ access_critical}$

An algorithm to **label** all states that satisfy EXp

We assume that all states satisfying p have been correctly labeled already

BuildEX(p) is

- 1 $\mathcal{Y} \leftarrow \{\mathbf{i} \in \mathcal{X}_{reach} : p \in labels(\mathbf{i})\};$
- 2 while $\mathcal{Y} \neq \emptyset$ do
- 3 pick and remove a state \mathbf{j} from \mathcal{Y} ;
- 4 for each $\mathbf{i} \in \mathcal{N}^{-1}(\mathbf{j})$ do
- 5 $labels(\mathbf{i}) \leftarrow labels(\mathbf{i}) \cup \{EXp\};$

initialize \mathcal{Y} with the states satisfying p

state \mathbf{i} can transition to state \mathbf{j}

An algorithm to **label** all states that satisfy $E[pUq]$

We assume that all states satisfying p and all states satisfying q have been correctly labeled already

BuildEU(p, q) is

1 $\mathcal{Y} \leftarrow \{\mathbf{i} \in \mathcal{Y}_{reach} : q \in labels(\mathbf{i})\};$

initialize \mathcal{Y} with the states satisfying q

2 for each $\mathbf{i} \in \mathcal{Y}$ do

3 $labels(\mathbf{i}) \leftarrow labels(\mathbf{i}) \cup \{E[pUq]\};$

4 while $\mathcal{Y} \neq \emptyset$ do

5 pick and remove a state \mathbf{j} from \mathcal{Y} ;

6 for each $\mathbf{i} \in \mathcal{N}^{-1}(\mathbf{j})$ do

state \mathbf{i} can transition to state \mathbf{j}

7 if $E[pUq] \notin labels(\mathbf{i})$ and $p \in labels(\mathbf{i})$ then

8 $labels(\mathbf{i}) \leftarrow labels(\mathbf{i}) \cup \{E[pUq]\};$

9 $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{\mathbf{i}\};$

An algorithm to **label** all states that satisfy EGp

We assume that all states satisfying p have been correctly labeled already

BuildEG(p) is

- 1 $\mathcal{Y} \leftarrow \{\mathbf{i} \in \mathcal{X}_{reach} : p \in labels(\mathbf{i})\};$ *initialize \mathcal{Y} with the states satisfying p*
- 2 build the set \mathcal{C} of SCCs in the subgraph of \mathcal{N} induced by \mathcal{Y} ;
- 3 $\mathcal{W} \leftarrow \{\mathbf{i} : \mathbf{i} \text{ is in a SCC of } \mathcal{C}\};$
- 4 for each $\mathbf{i} \in \mathcal{W}$ do
- 5 $labels(\mathbf{i}) \leftarrow labels(\mathbf{i}) \cup \{EGp\};$
- 6 while $\mathcal{W} \neq \emptyset$ do
- 7 pick and remove a state \mathbf{j} from \mathcal{W} ;
- 8 for each $\mathbf{i} \in \mathcal{N}^{-1}(\mathbf{j})$ do *state \mathbf{i} can transition to state \mathbf{j}*
- 9 if $EGp \notin labels(\mathbf{i})$ and $p \in labels(\mathbf{i})$ then
- 10 $labels(\mathbf{i}) \leftarrow labels(\mathbf{i}) \cup \{EGp\};$
- 11 $\mathcal{W} \leftarrow \mathcal{W} \cup \{\mathbf{i}\};$

This algorithm relies on finding the (nontrivial) strongly connected components (SCCs) of a graph

All sets of states and relations over sets of states are encoded using BDDs

An algorithm to build the BDD encoding the set of states that satisfy EXp

Assume that the BDD encoding the set \mathcal{P} of states satisfying p has been built already

BuildEXsymbolic(\mathcal{P}) is

1 return *RelationalProduct*($\mathcal{P}, \mathcal{N}^{-1}$);

perform one backward step in the transition relation

Two algorithms to build the BDD encoding the set of states that satisfy $E[pUq]$

Assume that the BDDs encoding the sets \mathcal{P} and \mathcal{Q} of states satisfying p and q have been built already

BuildEUsymbolic(\mathcal{P} , \mathcal{Q}) is

```
1  $\mathcal{Y} \leftarrow \emptyset;$ 
2  $\mathcal{U} \leftarrow \mathcal{Q};$            initialize the unexplored set  $\mathcal{U}$  with the states satisfying  $q$ 
3 repeat
4    $\mathcal{Y} \leftarrow \text{Union}(\mathcal{Y}, \mathcal{U});$            currently known states satisfying  $E[pUq]$ 
5    $\mathcal{W} \leftarrow \text{RelationalProduct}(\mathcal{U}, \mathcal{N}^{-1});$    perform one backward step in the transition relation
6    $\mathcal{Z} \leftarrow \text{Intersection}(\mathcal{W}, \mathcal{P});$            discard the states that do not satisfy  $p$ 
7    $\mathcal{U} \leftarrow \text{Difference}(\mathcal{Z}, \mathcal{Y});$            discard the states that are not new
8 until  $\mathcal{U} = \emptyset;$ 
9 return  $\mathcal{Y};$ 
```

BuildEUsymbolicAll(\mathcal{P} , \mathcal{Q}) is

```
1  $\mathcal{Y} \leftarrow \mathcal{Q};$            initialize the currently known result with the states satisfying  $q$ 
2 repeat
3    $\mathcal{O} \leftarrow \mathcal{Y};$            save the old set of states
4    $\mathcal{W} \leftarrow \text{RelationalProduct}(\mathcal{Y}, \mathcal{N}^{-1});$    perform one backward step in the transition relation
5    $\mathcal{Z} \leftarrow \text{Intersection}(\mathcal{W}, \mathcal{P});$            discard the states that do not satisfy  $p$ 
6    $\mathcal{Y} \leftarrow \text{Union}(\mathcal{Z}, \mathcal{Y});$            add to the currently known result
7 until  $\mathcal{O} = \mathcal{Y};$ 
8 return  $\mathcal{Y};$ 
```

An algorithm to build the BDD encoding the set of states that satisfy EGp

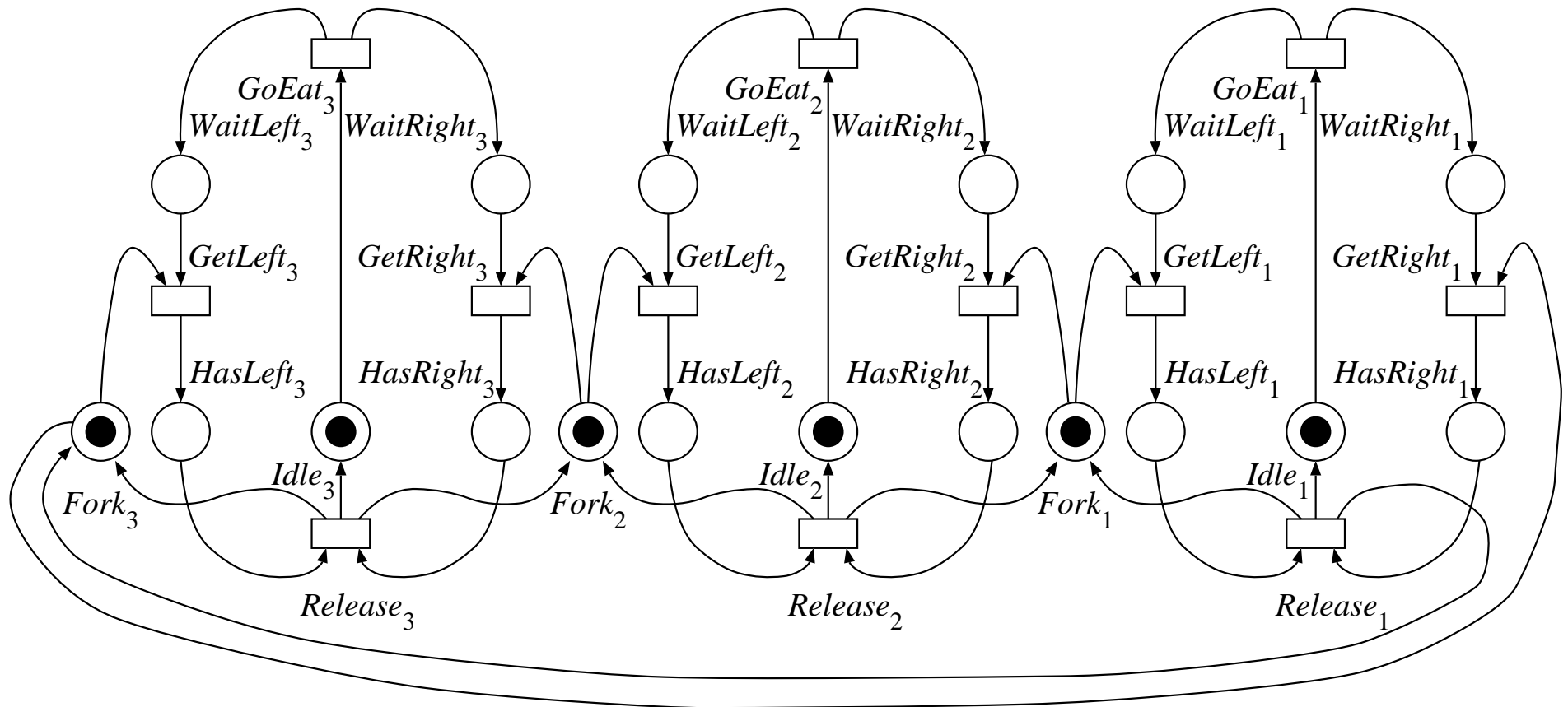
Assume that the BDDs encoding the set \mathcal{P} of states satisfying p has been built already

BuildEGsymbolic(\mathcal{P}) is

```
1  $\mathcal{Y} \leftarrow \mathcal{P};$  initialize  $\mathcal{Y}$  with the states satisfying  $p$ 
2 repeat
3    $\mathcal{O} \leftarrow \mathcal{Y};$  save the old set of states
4    $\mathcal{W} \leftarrow \text{RelationalProduct}(\mathcal{Y}, \mathcal{N}^{-1});$  perform one backward step in the transition relation
5    $\mathcal{Y} \leftarrow \text{Intersection}(\mathcal{Y}, \mathcal{W});$ 
6 until  $\mathcal{O} = \mathcal{Y};$ 
7 return  $\mathcal{Y};$ 
```

This algorithm starts with a larger set of states and **reduces** it

This algorithm is **not** based on finding the strongly connected components of \mathcal{N}



N subnets connected in a circular fashion

```
spn phils(int N) := {
  for (int i in {0..N-1}) {
    place Idle[i], WaitL[i], WaitR[i], HasL[i], HasR[i], Fork[i];
    partition(i+1:Idle[i]:WaitL[i]:WaitR[i]:HasL[i]:HasR[i]:Fork[i]);
    trans GoEat[i], GetL[i], GetR[i], Release[i];
    init(Idle[i]:1, Fork[i]:1);
  }
  for (int i in {0..N-1}) {
    arcs(Idle[i]:GoEat[i], GoEat[i]:WaitL[i], GoEat[i]:WaitR[i],
        WaitL[i]:GetL[i], Fork[i]:GetL[i], GetL[i]:HasL[i],
        WaitR[i]:GetR[i], Fork[mod(i+1, N)]:GetR[i], GetR[i]:HasR[i],
        HasL[i]:Release[i], HasR[i]:Release[i], Release[i]:Idle[i],
        Release[i]:Fork[i], Release[i]:Fork[mod(i+1, N)]);
  }
  bigint num := card(reachable);
  stateset g := EF(initialstate);          bigint numg := card(g);
  stateset b := difference(reachable,g);    bool out      := printset(b);
};
# StateStorage MDD_SATURATION
int N := read_int("number of philosophers"); print("N=",N,"\n");
print("Reachable states: ",phils(N).num,"\n");
print("Good states:      ",phils(N).numg,"\n");
print("The bad states are:"); phils(N).out;
```

Reading input.

N=50

Reachable states: 22,291,846,172,619,859,445,381,409,012,498

Good states: 22,291,846,172,619,859,445,381,409,012,496

The bad states are:

State 0 : { WaitR[0]:1 HasL[0]:1 WaitR[1]:1 HasL[1]:1 WaitR[2]:1 HasL[2]:1 WaitL[2]:1

State 1 : { WaitL[0]:1 HasR[0]:1 WaitL[1]:1 HasR[1]:1 WaitL[2]:1 HasR[2]:1 WaitR[2]:1

true

Done.

Decision diagrams for integer functions

Assume a **domain** $\mathcal{X}_{pot} = \mathcal{X}_L \times \cdots \times \mathcal{X}_1$, where $\mathcal{X}_k = \{0, 1, \dots, n_k - 1\}$, for some $n_k \in \mathbb{N}$

Assume a **range** $\mathcal{X}_0 = \{0, 1, \dots, n_0 - 1\}$, for some $n_0 \in \mathbb{N}$ (or an arbitrary \mathcal{X}_0 ...)

An **MTMDD** is an acyclic directed edge-labeled graph where:

- The only **terminal** nodes are values from \mathcal{X}_0 and are at **level 0** $\forall i_0 \in \mathcal{X}_0, i_0.lvl = 0$
- A **nonterminal** node p is at a **level** k , with $L \geq k \geq 1$ $p.lvl = k$
- A nonterminal node p at level k has n_k outgoing edges pointing to **children** $p[i_k]$, for $i_k \in \mathcal{X}_k$
- The level of the children is lower than that of p ; $p[0].lvl < p.lvl, p[1].lvl < p.lvl$
- A node p at level k encodes the **function** $v_p : \mathcal{X}_{pot} \rightarrow \mathcal{X}_0$ defined recursively by

$$v_p(x_L, \dots, x_1) = \begin{cases} p & \text{if } k = 0 \\ v_{p[x_k]}(x_L, \dots, x_1) & \text{if } k > 0 \end{cases}$$

Instead of levels, we can also talk of **variables**:

- The terminal nodes are associated with the **range variable** x_0
- A nonterminal node is associated with a **domain variable** x_k , with $L \geq k \geq 1$

For **canonical** MTMDDs, we further require that

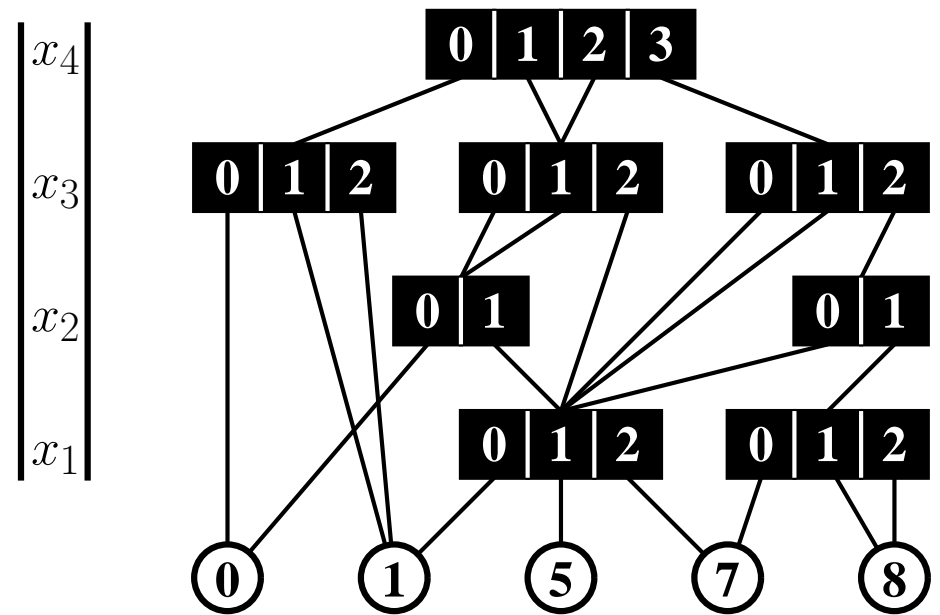
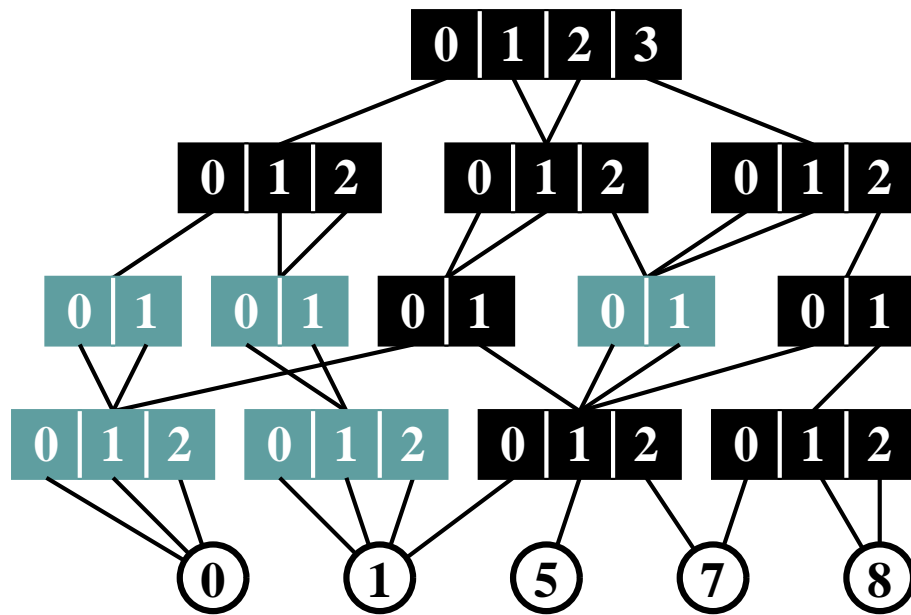
- There are no **duplicates**: if $p.lvl = q.lvl = k$ and $p[i_k] = q[i_k]$ for all $i_k \in \mathcal{X}_k$, then $p = q$

Then, if the MTMDD is **quasi-reduced**, there is **no level skipping**:

- The only **root** nodes with no incoming arcs are at level L
- Each child $p[i_k]$ of a node p is at level $p.lvl - 1$

Or, if the MTMDD is **fully-reduced**, there is **maximum level skipping**:

- There are no **redundant** nodes p satisfying $p[i_k] = q$ for all $i_k \in \mathcal{X}_k$



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

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

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

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

These MTMDDs encode a function $\mathcal{X}_{pot} = \mathcal{X}_4 \times \cdots \times \mathcal{X}_1 \rightarrow \mathbb{N}$

Assume a **domain** $\mathcal{X}_{pot} = \mathcal{X}_L \times \cdots \times \mathcal{X}_1$, where $\mathcal{X}_k = \{0, 1, \dots, n_k - 1\}$, for some $n_k \in \mathbb{N}$

Assume the **range** \mathbb{Z}

An EVMDD is an acyclic directed edge-labeled graph where:

- The only **terminal** node is Ω and is at **level 0** $\Omega.lvl = 0$
- A **nonterminal** node p is at a **level** k , with $L \geq k \geq 1$ $p.lvl = k$
- A nonterminal node p at level k has n_k outgoing edges
- For $i_k \in \mathcal{X}_k$, edge $p[i_k]$ points to **child** $p[i_k].child$, and has **value** $p[i_k].val \in \mathbb{Z}$
- The level of the children is lower than that of p $p[i_k].child.lvl < p.lvl$
- An edge $\langle \sigma, p \rangle$, with $p.lvl = k$ encodes the **function** $v_{\langle \sigma, p \rangle} : \mathcal{X}_{pot} \rightarrow \mathbb{Z}$ defined recursively by

$$v_{\langle \sigma, p \rangle}(x_L, \dots, x_1) = \begin{cases} \sigma & \text{if } k = 0 \\ \sigma + v_{p[x_k]}(x_L, \dots, x_1) & \text{if } k > 0 \end{cases}$$

For **canonical** EVMDDs, we first **normalize** each node p at level $k \geq 1$ in one of two ways:

- $p[0].val = 0$, or EVMDDs
- $p[i_k].val \geq 0$ for all $i_k \in \mathcal{X}_k$, and $p[j_k] = 0$ for at least one $j_k \in \mathcal{X}_k$ EV⁺MDDs

Then, the usual reduction requirements apply:

- There are no **duplicates**: if $p.lvl = q.lvl = k$ and $p[i_k] = q[i_k]$ for all $i_k \in \mathcal{X}_k$, then $p = q$

And, if the MDD is **quasi-reduced**, there is **no level skipping**:

- The only **root** nodes with no incoming arcs are at level L , and have **root edge values** in \mathbb{Z}
- Each child $p[i_k].child$ of a node p is at level $p.lvl - 1$

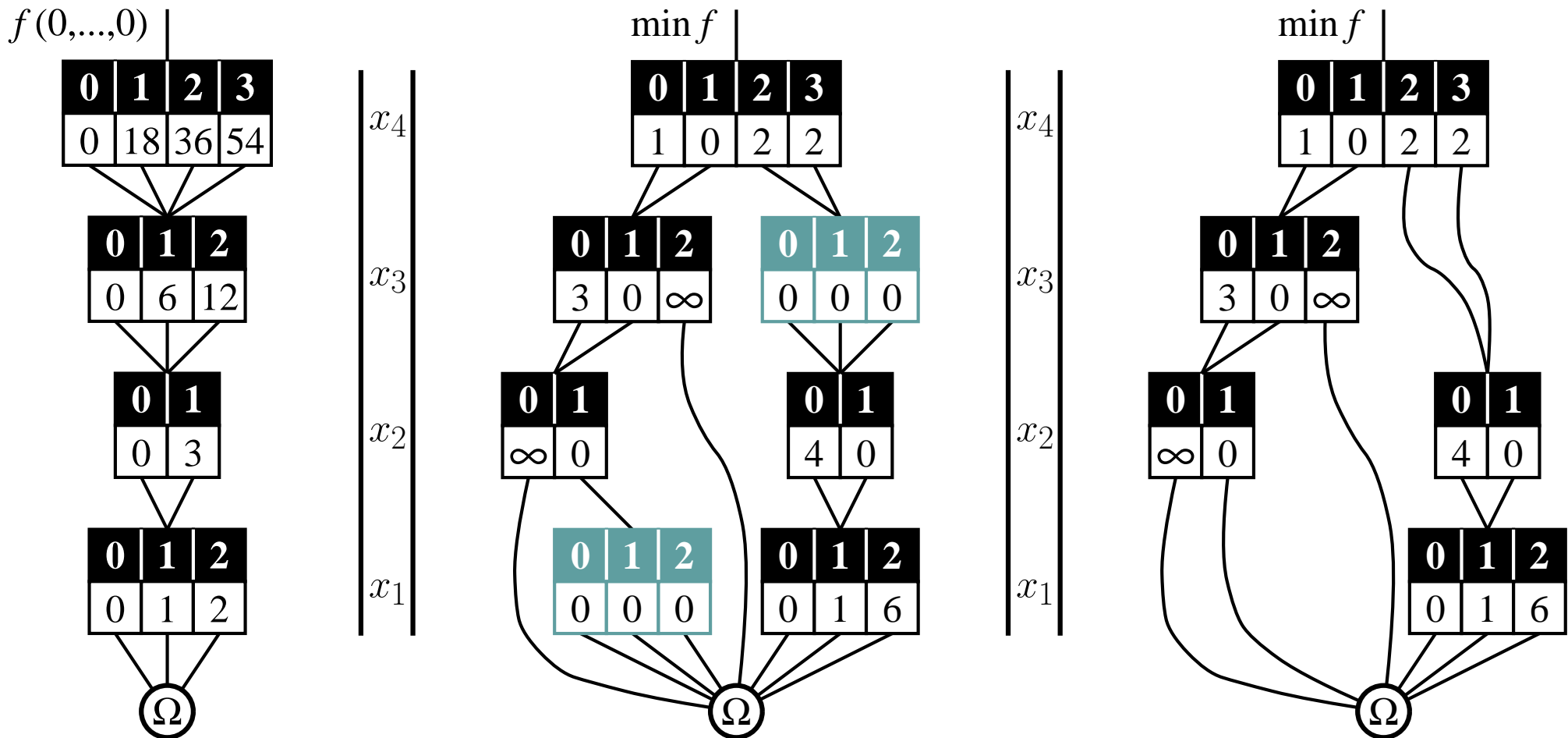
Or, if the MDD is **fully-reduced**, there is **maximum level skipping**:

- There are no **redundant** nodes p satisfying $p[i_k].child = q$ and $p[i_k].val = 0$ for all $i_k \in \mathcal{X}_k$

For EVMDDs, the value of the incoming root edge is $f(0, \dots, 0)$

For EV⁺MDDs, the value of the incoming root edge is $\min f$

The EV⁺MDDs normalization allows to store **partial functions** $\mathcal{X}_{pot} \rightarrow \mathbb{Z} \cup \{\infty\}$

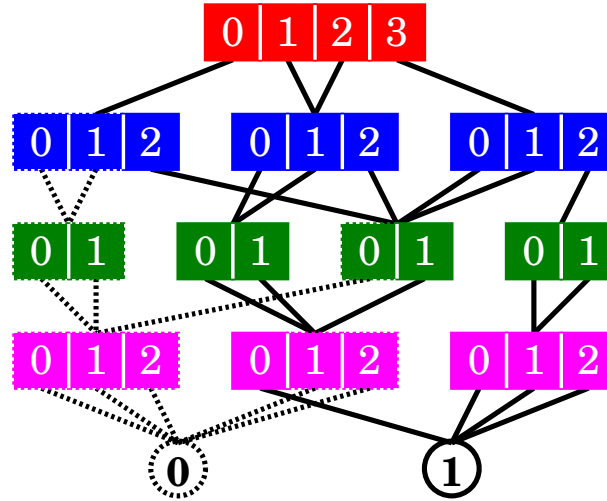


$$\mathcal{X}_4 = \{0, 1, 2, 3\}$$

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

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

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



$$\mathcal{Y} = \left\{ \begin{array}{cccccccccccccccccccc} 0 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 0 & 0 & 1 & 1 & 2 & 0 & 0 & 1 & 1 & 2 & 0 & 1 & 2 & 2 & 2 & 2 & 2 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 1 & 2 \end{array} \right\}$$

To compute the index of a state, use edge values:

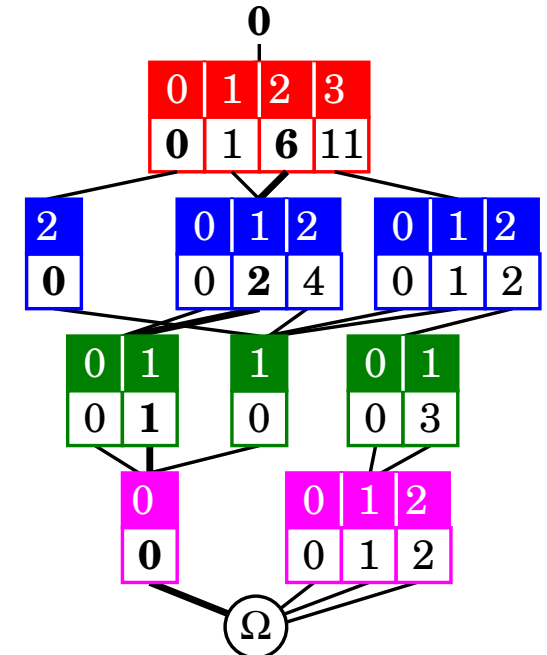
- Sum the values found on the corresponding path:

$$\psi(2, 1, 1, 0) = 0 + 6 + 2 + 1 + 0 = 9$$

- A state is unreachable if the path is not complete:

$$\psi(0, 2, 0, 0) = 0 + 0 + 0 + \infty = \infty$$

(a missing edge has the default value of ∞)



Given a model $(\mathcal{X}_{pot}, \mathcal{X}_{init}, \mathcal{N})$, we can define the **distance** function $\delta : \mathcal{X}_{pot} \rightarrow \mathbb{N} \cup \{\infty\}$

$$\delta(\mathbf{i}) = \min\{d : \mathbf{i} \in \mathcal{N}^d(\mathcal{X}_{init})\}$$

$$\text{thus } \delta(\mathbf{i}) = \infty \Leftrightarrow \mathbf{i} \notin \mathcal{X}_{reach}$$

Build $\mathcal{X}^{[d]} = \{\mathbf{i} : \delta(\mathbf{i}) = d\}$,

for $d = 0, 1, \dots, d_{max}$

Build $\mathcal{Y}^{[d]} = \{\mathbf{i} : \delta(\mathbf{i}) \leq d\}$,

for $d = 0, 1, \dots, d_{max}$

DistanceMddForestEQ $(\mathcal{X}_{init}, \mathcal{N})$ is

```
1  $d \leftarrow 0$ ;  
2  $\mathcal{X}_{reach} \leftarrow \mathcal{X}_{init}$ ;  
3  $\mathcal{X}^{[0]} \leftarrow \mathcal{X}_{init}$ ;  
4 repeat  
5    $\mathcal{X}^{[d+1]} \leftarrow \mathcal{N}(\mathcal{X}^{[d]}) \setminus \mathcal{X}_{reach}$ ;  
6    $d \leftarrow d + 1$ ;  
7    $\mathcal{X}_{reach} \leftarrow \mathcal{X}_{reach} \cup \mathcal{X}^{[d]}$ ;  
8 until  $\mathcal{X}^{[d]} = \emptyset$ ;
```

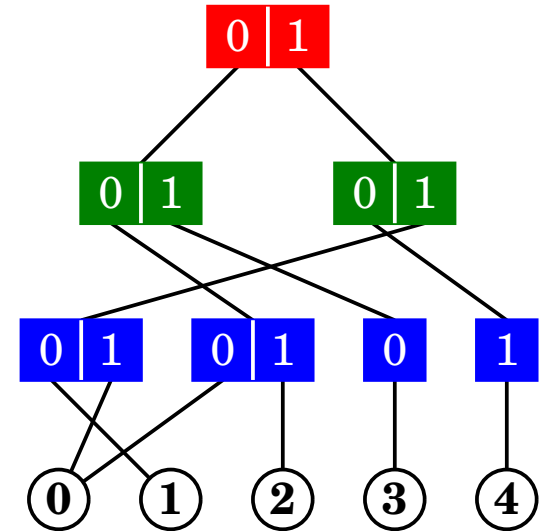
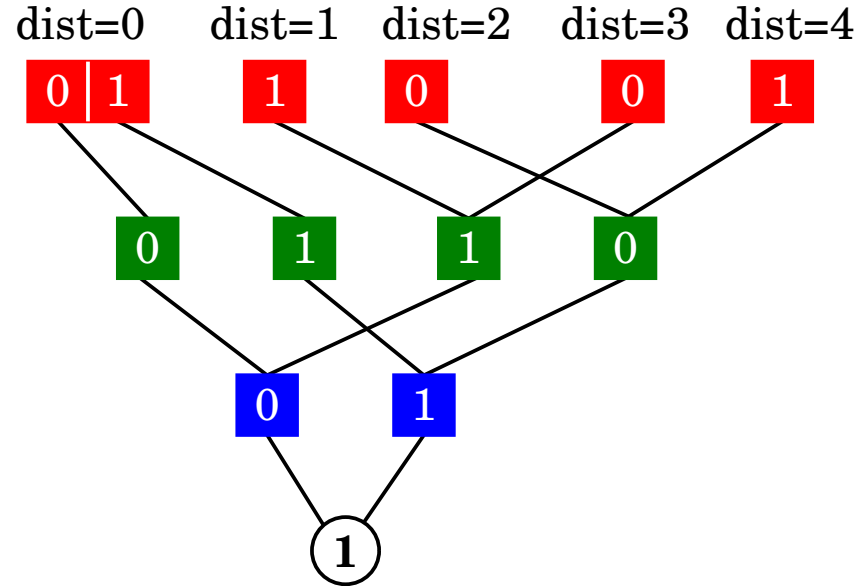
DistanceMddForestLE $(\mathcal{X}_{init}, \mathcal{N})$ is

```
1  $d \leftarrow 0$ ;  
2  $\mathcal{Y}^{[0]} \leftarrow \mathcal{X}_{init}$ ;  
3 repeat  
4    $\mathcal{Y}^{[d+1]} \leftarrow \mathcal{N}(\mathcal{Y}^{[d]}) \cup \mathcal{Y}^{[d]}$ ;  
5    $d \leftarrow d + 1$ ;  
6 until  $\mathcal{Y}^{[d]} = \mathcal{Y}^{[d-1]}$ ;
```

This is breadth-first symbolic state space generation

$\mathcal{X}_{reach} = \{\mathbf{i} \in \mathcal{X}_{pot} : \delta(\mathbf{i}) < \infty\} = \bigcup_{d=0}^{d_{max}} \mathcal{X}^{[d]} = \mathcal{Y}^{[d_{max}]}$ is a by-product of this process!

x_3	0 0 0 0	1 1 1 1
x_2	0 0 1 1	0 0 1 1
x_1	0 1 0 1	0 1 0 1
f	0 2 3 ∞ ∞	4 1 0

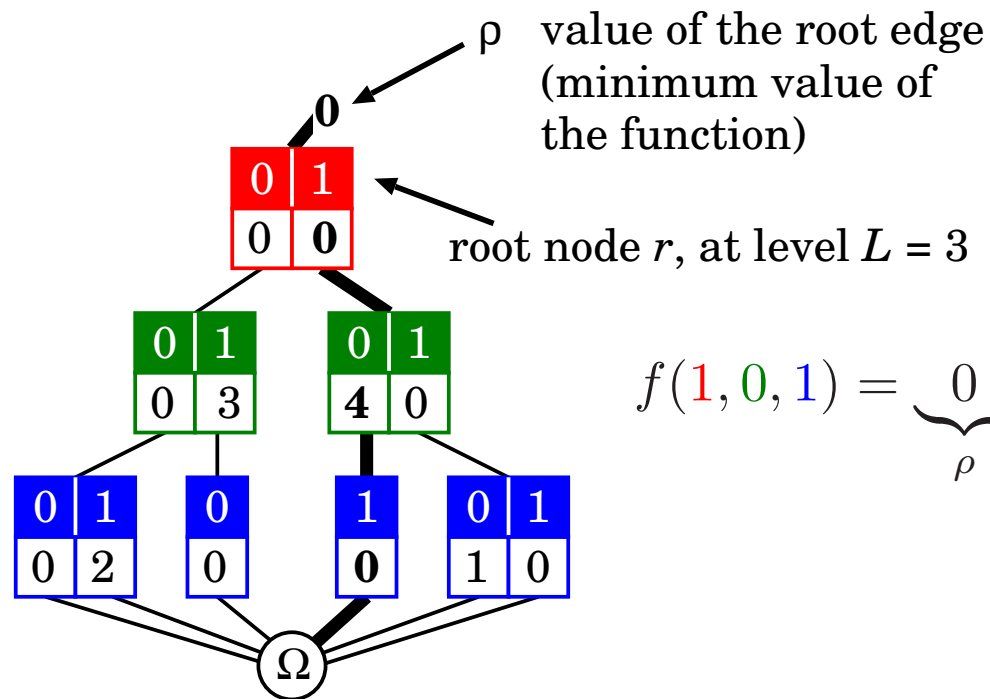


With an MDD forest: node merging can be poor at the top

With an MTMDD: node merging can be poor at the bottom

Both approaches are explicit in the number of distinct distance values

x_3	0	0	0	0	1	1	1	1
x_2	0	0	1	1	0	0	1	1
x_1	0	1	0	1	0	1	0	1
f	0	2	3	∞	∞	4	1	0



$$f(1, 0, 1) = \underbrace{0}_{\rho} + (0 + 4 + 0) = 4$$

If $\langle \rho, r \rangle$ encodes f , then $\rho = \min\{f(\mathbf{i}) : \mathbf{i} \in \mathcal{X}_{pot}\}$

EV^+ MDDs can canonically represent all functions $\mathcal{X}_{pot} \rightarrow \mathbb{Z} \cup \{\infty\}$

EVBDDs [Lai et al. 1992] cannot represent certain partial functions

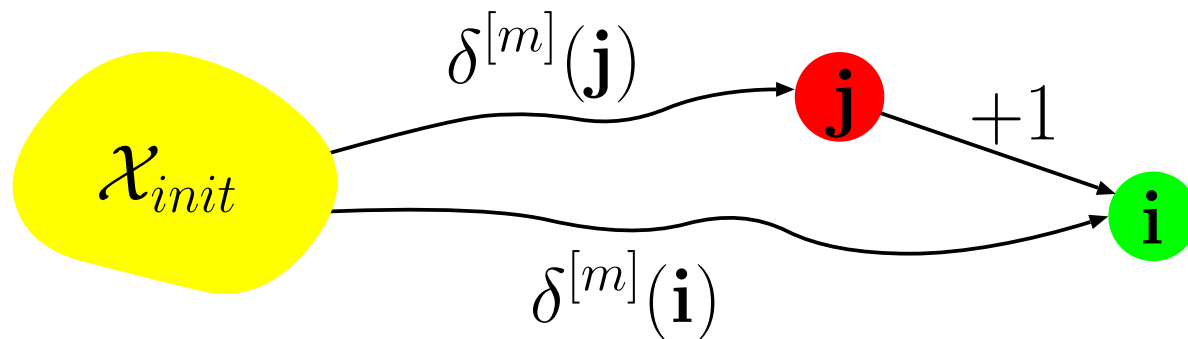
It is easy to build the distance function $\delta : \mathcal{X}_{pot} \rightarrow \mathbb{N} \cup \{\infty\}$ using a **breadth-first** iteration

$$\delta(\mathbf{i}) = \min\{d : \mathbf{i} \in \mathcal{N}^d(\mathcal{X}_{init})\} \quad \text{thus} \quad \delta(\mathbf{i}) = \infty \Leftrightarrow \mathbf{i} \notin \mathcal{X}_{reach}$$

To use **Saturation** instead, think of δ as the **fixed-point** of the iteration $\delta^{[m+1]} = \Phi(\delta^{[m]})$ where

$$\delta^{[m+1]}(\mathbf{i}) = \min \left(\delta^{[m]}(\mathbf{i}), \min \left\{ 1 + \delta^{[m]}(\mathbf{j}) \mid \exists \alpha \in \mathcal{E} : \mathbf{i} \in \mathcal{N}_\alpha(\mathbf{j}) \right\} \right)$$

initialized with $\delta^{[0]}(\mathbf{i}) = 0$ if $\mathbf{i} \in \mathcal{X}_{init}$ and $\delta^{[0]}(\mathbf{i}) = \infty$ otherwise



edge *Minimum*(level k , edge $\langle \alpha, p \rangle$, edge $\langle \beta, q \rangle$)

edge is a pair $\langle int, node \rangle$

local node p', q', r ;

local int μ, α', β' ;

local local i_k ;

1 if $\alpha = \infty$ then return $\langle \beta, q \rangle$;

2 if $\beta = \infty$ then return $\langle \alpha, p \rangle$;

3 $\mu \leftarrow \min\{\alpha, \beta\}$;

4 if $k = 0$ then return $\langle \mu, \Omega \rangle$;

the only node at level 0 is Ω

5 if *Cache* contains entry $\langle \text{MinimumCODE}, k, p, q, \alpha - \beta : \gamma, r \rangle$ then return $\langle \gamma + \mu, r \rangle$;

6 $r \leftarrow \text{NewNode}(k)$;

create new node at level k with edges set to $\langle \infty, \Omega \rangle$

7 foreach $i_k \in \mathcal{X}_k$ do

8 $p' \leftarrow p.\text{child}[i_k]$;

9 $\alpha' \leftarrow \alpha - \mu + p.\text{val}[i_k]$;

10 $q' \leftarrow q.\text{child}[i_k]$;

11 $\beta' \leftarrow \beta - \mu + q.\text{val}[i_k]$;

12 $r[i_k] \leftarrow \text{Minimum}(k-1, \langle \alpha', p' \rangle, \langle \beta', q' \rangle)$;

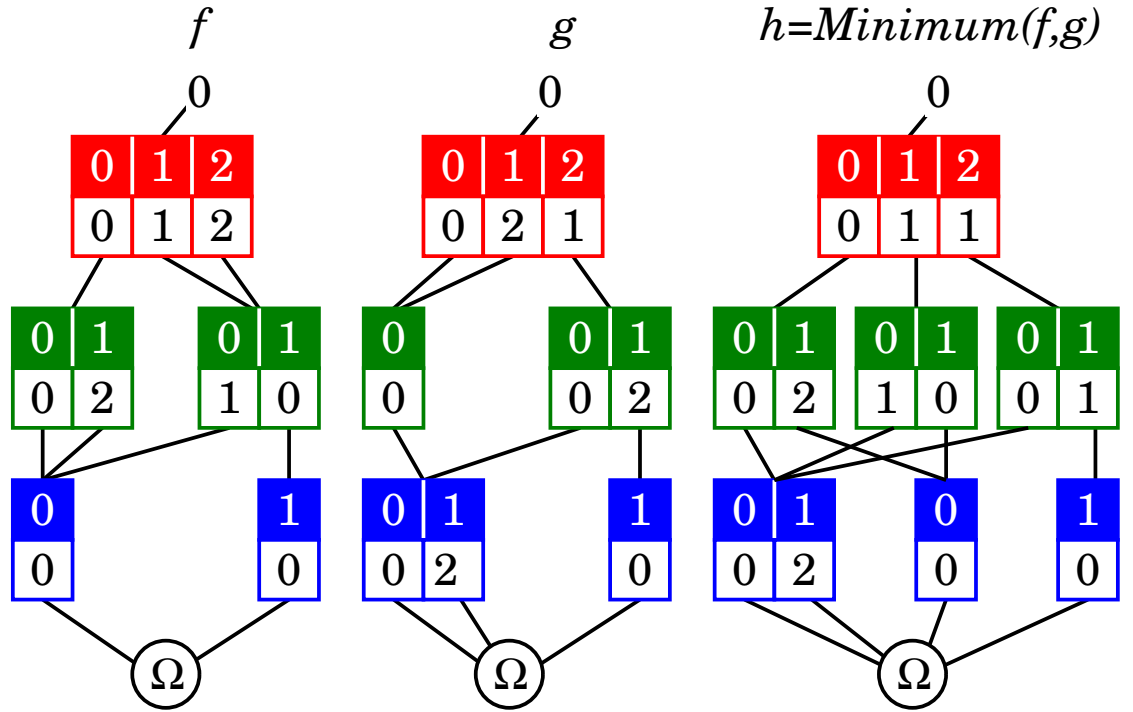
continue downstream

13 *UniqueTableInsert*(k, r);

14 enter $\langle \text{MinimumCODE}, k, p, q, \alpha - \beta : \mu, r \rangle$ in *Cache*;

15 return $\langle \mu, r \rangle$;

x_3	0	0	0	0	1	1	1	1	2	2	2	2
x_2	0	0	1	1	0	0	1	1	0	0	1	1
x_1	0	1	0	1	0	1	0	1	0	1	0	1
f	0	∞	2	∞	2	∞	∞	1	3	∞	∞	2
g	0	2	∞	∞	2	4	∞	∞	1	3	∞	3
h	0	2	2	∞	2	4	∞	1	1	3	∞	2



N	$ S $	Time (in seconds)					Final nodes				Peak nodes				
		E_s	E_b	M_b	T_s	T_b	E_s	E_b	M_b	T_s	T_b	E_s	E_b	M_b	T_s

Dining philosophers: $d_{max} = 2N$, $L = N/2$, $|\mathcal{X}_k| = 34$ for all k

10	1.9×10^6	0.01	0.06	0.05	0.12	0.46	21	255	170	21	605	644	238	4022
30	6.4×10^{18}	0.02	0.86	0.70	7.39	56.80	71	2545	1710	71	7225	7364	2788	140262
1000	9.2×10^{626}	0.48	—	—	—	—	2496	—	—	2496	—	—	—	—

Kanban system: $d_{max} = 14N$, $L = 4$, $|\mathcal{X}_k| = (N+3)(N+2)(N+1)/6$ for all k

5	2.5×10^6	0.02	0.14	0.12	0.24	1.55	9	444	133	57	1132	1156	776	13241
12	5.5×10^9	0.34	4.34	3.45	11.08	129.46	16	2368	518	218	5633	5805	5585	165938
50	1.0×10^{16}	179.48	—	—	—	—	58	—	—	2802	—	—	—	—

Flex. manuf. syst.: $d_{max} = 14N$, $L = 19$, $|\mathcal{X}_k| = N+1$ for all k except $|\mathcal{X}_{17}| = 4$, $|\mathcal{X}_{12}| = 3$, $|\mathcal{X}_2| = 2$

5	2.9×10^6	0.01	0.42	0.34	0.88	11.78	149	5640	2989	211	15205	15693	4903	179577
10	2.5×10^9	0.04	2.96	2.40	5.79	608.92	354	28225	11894	536	76676	78649	17885	1681625
140	2.0×10^{23}	20.03	—	—	—	—	32012	—	—	52864	—	—	—	—

Round-robin mutex protocol: $d_{max} = 8N - 6$, $L = N + 1$, $|\mathcal{X}_k| = 10$ for all k except $|\mathcal{X}_1| = N + 1$

10	2.3×10^4	0.01	0.06	0.05	0.22	0.50	92	1038	1123	107	1898	1948	1210	9245
30	7.2×10^{10}	0.05	0.95	0.89	16.04	224.83	582	12798	19495	637	24122	24566	20072	376609
200	7.2×10^{62}	1.63	—	—	—	—	20897	—	—	21292	—	—	—	—

E_s : EV⁺MDD & Saturation E_b : EV⁺MDD & breadth-first M_b : multiple MDDs & breadth-first

T_s : MTMDD & Saturation T_b : MTMDD & breadth-first

INPUT: the MDD x encoding a set of states \mathcal{X} , the EV⁺MDD $\langle \rho, r \rangle$ encoding δ

OUTPUT: a (minimum) μ -length trace $\mathbf{j}^{[0]}, \dots, \mathbf{j}^{[\mu]}$ from a state in \mathcal{X}_{init} to a state in \mathcal{X}

1. Build the EV⁺MDD $\langle 0, x \rangle$ encoding $\delta_x(\mathbf{i}) = 0$ if $\mathbf{i} \in \mathcal{X}$ and $\delta_x(\mathbf{i}) = \infty$ if $\mathbf{i} \in \mathcal{X}_{pot} \setminus \mathcal{X}$
2. Compute the EV⁺MDD $\langle \mu, m \rangle$ encoding $Max(\langle \rho, r, \rangle \langle 0, x \rangle)$
 μ is the length of one of the shortest-paths we are seeking
3. If $\mu = \infty$, exit: \mathcal{X} does not contain reachable states
4. Otherwise, extract from $\langle \mu, m \rangle$ a state $\mathbf{j}^{[\mu]} = (j_L^{[\mu]}, \dots, j_1^{[\mu]})$ on a 0-labelled path from m to Ω
 $\mathbf{j}^{[\mu]}$ is a reachable state in \mathcal{X} at the desired minimum distance μ from \mathcal{X}_{init}
5. Initialize ν to μ and iterate until $\nu = 0$:
 - (a) For each state $\mathbf{i} \in \mathcal{X}_{pot}$ such that $\mathbf{j}^{[\nu]} \in \mathcal{N}(\mathbf{i})$ (use the backward function \mathcal{N}^{-1})
 - compute $\delta(\mathbf{i})$ using $\langle \rho, r \rangle$ and stop on the first \mathbf{i} such that $\delta(\mathbf{i}) = \nu - 1$
there exists at least one such state \mathbf{i}^*
 - (b) Decrement ν
 - (c) Let $\mathbf{j}^{[\nu]}$ be \mathbf{i}^*

Decision diagrams for Markov models

A stochastic process $\{X(t) : t \geq 0\}$ is a collection of r.v.'s indexed by a time parameter t

We say that $X(t)$ is the **state** of the process at time t

The possible values $X(t)$ can ever assume for any t is (a subset of) the **state space** \mathcal{X}_{reach}

$\{X(t) : t \geq 0\}$ over a discrete \mathcal{X}_{reach} is a continuous-time Markov chain (CTMC) if

$$\begin{aligned} \Pr \left\{ X(t^{[n+1]}) = \mathbf{i}^{[n+1]} \mid X(t^{[n]}) = \mathbf{i}^{[n]} \wedge X(t^{[n-1]}) = \mathbf{i}^{[n-1]} \wedge \dots \wedge X(t^{[0]}) = \mathbf{i}^{[0]} \right\} \\ = \Pr \left\{ X(t^{[n+1]}) = \mathbf{i}^{[n+1]} \mid X(t^{[n]}) = \mathbf{i}^{[n]} \right\} \end{aligned}$$

for any $0 \leq t^{[0]} \leq \dots \leq t^{[n-1]} \leq t^{[n]} \leq t^{[n+1]}$ and $\mathbf{i}^{[0]}, \dots, \mathbf{i}^{[n-1]}, \mathbf{i}^{[n]}, \mathbf{i}^{[n+1]} \in \mathcal{X}_{reach}$

Markov property:

“given the present state, the future is independent of the past”

“the most recent knowledge about the state is all we need”

A continuous-time Markov chain (CTMC) $\{X(t) : t \geq 0\}$ with state space \mathcal{X}_{reach} is described by

- its infinitesimal generator $\mathbf{Q} = \mathbf{R} - \text{diag}(\mathbf{R} \cdot \mathbf{1}) = \mathbf{R} - \text{diag}(\mathbf{h})^{-1} \in \mathbb{R}^{|\mathcal{X}_{reach}| \times |\mathcal{X}_{reach}|}$
- its initial probability vector $\boldsymbol{\pi}(0) \in \mathbb{R}^{|\mathcal{X}_{reach}|}$

where

- \mathbf{R} is the transition rate matrix: $\mathbf{R}[\mathbf{i}, \mathbf{j}]$ is the rate of going to state \mathbf{j} when in state \mathbf{i}
- \mathbf{h} is the expected holding time vector: $\mathbf{h}[\mathbf{i}] = 1 / \sum_{\mathbf{j} \in \mathcal{X}_{reach}} \mathbf{R}[\mathbf{i}, \mathbf{j}]$
- $\boldsymbol{\pi}(0)[\mathbf{i}] = \Pr \{\text{chain is in state } \mathbf{i} \text{ at time } 0, \text{ i.e., initially}\}$

Transient probability vector $\boldsymbol{\pi}(t) \in \mathbb{R}^{|\mathcal{X}_{reach}|}$: $\boldsymbol{\pi}(t)[\mathbf{i}] = \Pr \{X(t) = \mathbf{i}\}$

- $\boldsymbol{\pi}(t)$ is the solution of $\frac{d\boldsymbol{\pi}(t)}{dt} = \boldsymbol{\pi}(t) \cdot \mathbf{Q}$ with initial condition $\boldsymbol{\pi}(0)$

Steady-state probability vector $\boldsymbol{\pi} \in \mathbb{R}^{|\mathcal{X}_{reach}|}$: $\boldsymbol{\pi}[\mathbf{i}] = \lim_{t \rightarrow \infty} \Pr \{X(t) = \mathbf{i}\}$

- $\boldsymbol{\pi}$ is the solution of $\boldsymbol{\pi} \cdot \mathbf{Q} = \mathbf{0}$ subject to $\sum_{\mathbf{i} \in \mathcal{X}_{reach}} \boldsymbol{\pi}[\mathbf{i}] = 1$ \mathbf{Q} must be ergodic

For ergodic CTMCs: solve $\boldsymbol{\pi} \cdot \mathbf{Q} = \mathbf{0}$ subject to $\sum_{\mathbf{i} \in \mathcal{X}_{reach}} \boldsymbol{\pi}[\mathbf{i}] = 1$ $rank(\mathbf{Q}) = |\mathcal{X}_{reach}| - 1$

Direct methods are rarely applicable in practice Iterative methods are preferable as they avoid fill-in

Jacobi(in: $\boldsymbol{\pi}^{(old)}$, \mathbf{h} , \mathbf{R} ; out: $\boldsymbol{\pi}^{(new)}$) is

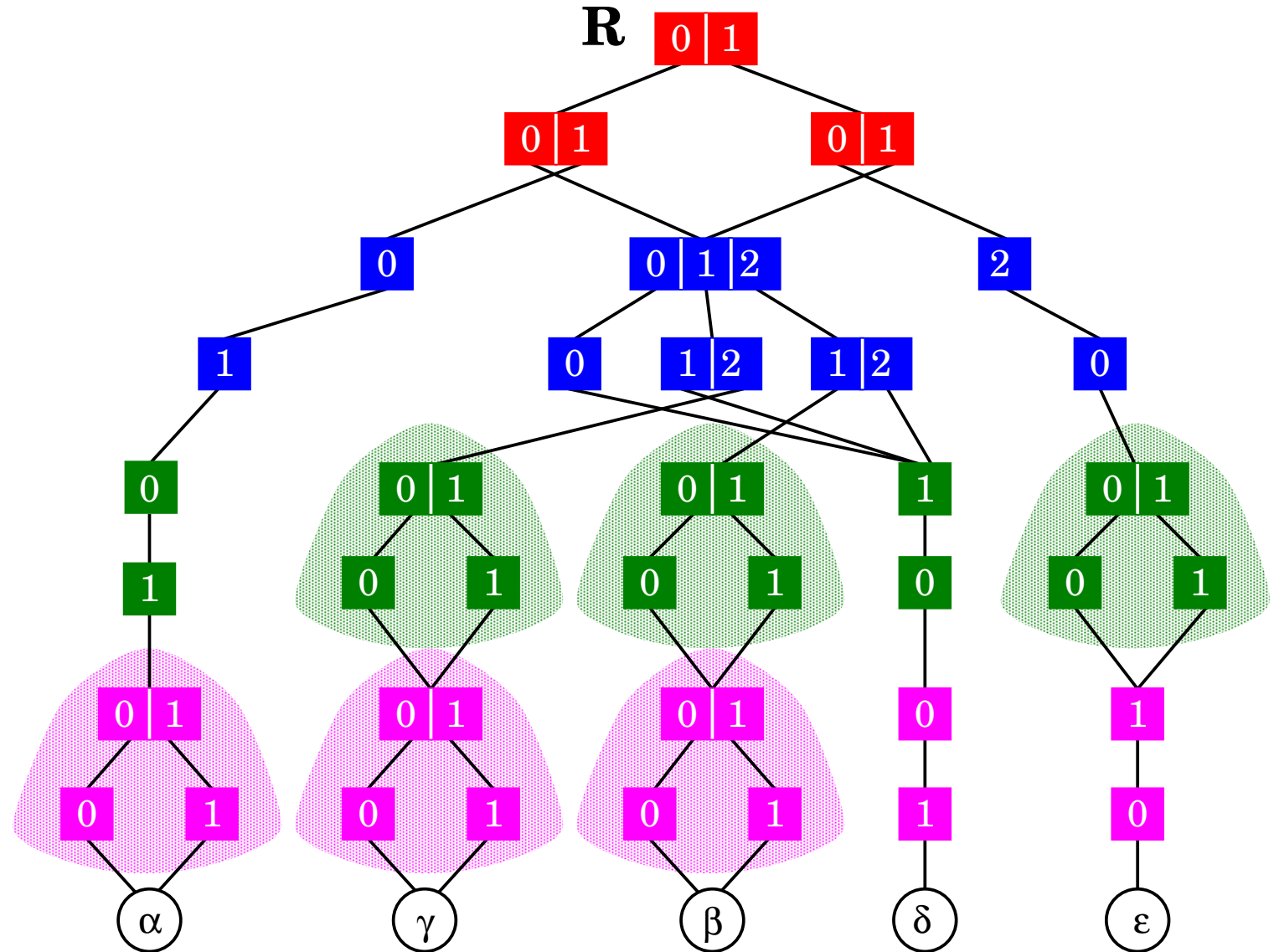
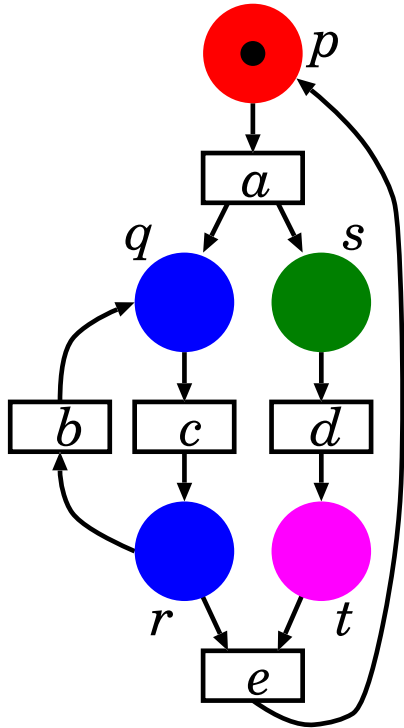
```
1  repeat
2    for  $\mathbf{j} = 0$  to  $|\mathcal{X}_{reach}| - 1$ 
3       $\boldsymbol{\pi}^{(new)}[\mathbf{j}] \leftarrow \mathbf{h}[\mathbf{j}] \cdot \sum_{0 \leq \mathbf{i} < |\mathcal{X}_{reach}|, \mathbf{i} \neq \mathbf{j}} \boldsymbol{\pi}^{(old)}[\mathbf{i}] \cdot \mathbf{R}[\mathbf{i}, \mathbf{j}]$ ;
4      “renormalize  $\boldsymbol{\pi}^{(new)}$ ”;
5       $\boldsymbol{\pi}^{(old)} \leftarrow \boldsymbol{\pi}^{(new)}$ ;
6  until “converged”;
7  return  $\boldsymbol{\pi}^{(new)}$ ;
```

GaussSeidel(in: \mathbf{h} , \mathbf{R} ; inout: $\boldsymbol{\pi}$) is

```
1  repeat
2    for  $\mathbf{j} = 0$  to  $|\mathcal{X}_{reach}| - 1$ 
3       $\boldsymbol{\pi}[\mathbf{j}] \leftarrow \mathbf{h}[\mathbf{j}] \cdot \sum_{0 \leq \mathbf{i} < |\mathcal{X}_{reach}|, \mathbf{i} \neq \mathbf{j}} \boldsymbol{\pi}[\mathbf{i}] \cdot \mathbf{R}[\mathbf{i}, \mathbf{j}]$ ;
4      “renormalize  $\boldsymbol{\pi}$ ”;
5  until “converged”;
6  return  $\boldsymbol{\pi}$ ;
```

Gauss-Seidel converges faster than Jacobi but requires strict **by-column** access to the entries of \mathbf{R}

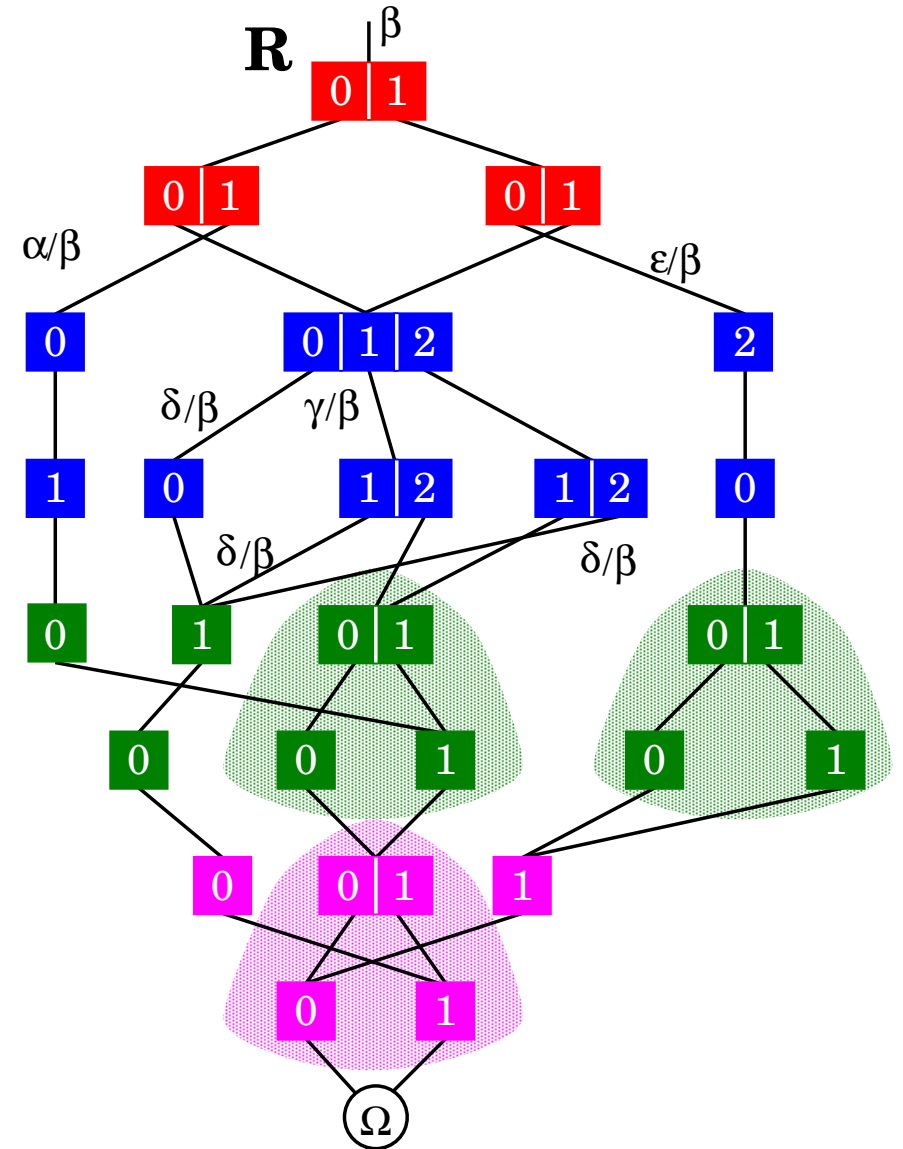
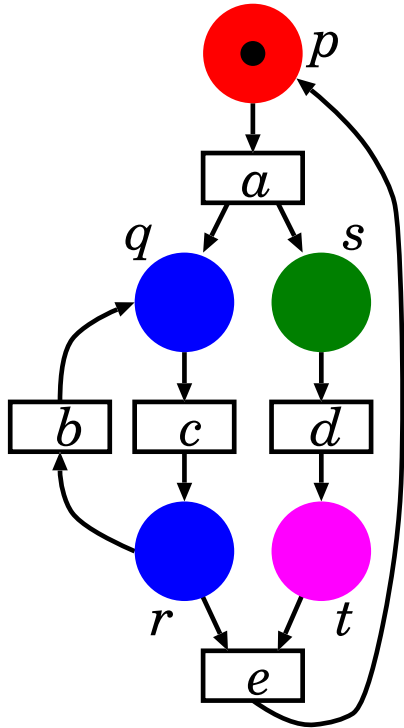
$$\mathcal{X}_4: \{p^1, p^0\} \equiv \{0, 1\} \quad \mathcal{X}_3: \{q^0 r^0, q^1 r^0, q^0 r^1\} \equiv \{0, 1, 2\} \quad \mathcal{X}_2: \{s^0, s^1\} \equiv \{0, 1\} \quad \mathcal{X}_1: \{t^0, t^1\} \equiv \{0, 1\}$$



α = rate of a
 β = rate of b
 γ = rate of c
 δ = rate of d
 ϵ = rate of e

note the shaded identity patterns!!!

$$\mathcal{X}_4: \{p^1, p^0\} \equiv \{0, 1\} \quad \mathcal{X}_3: \{q^0 r^0, q^1 r^0, q^0 r^1\} \equiv \{0, 1, 2\} \quad \mathcal{X}_2: \{s^0, s^1\} \equiv \{0, 1\} \quad \mathcal{X}_1: \{t^0, t^1\} \equiv \{0, 1\}$$



α = rate of a
 β = rate of b
 γ = rate of c
 δ = rate of d
 ϵ = rate of e

identity patterns remain!!!

Assume a **domain** $\mathcal{X}_{pot} = \mathcal{X}_L \times \cdots \times \mathcal{X}_1$, where $\mathcal{X}_k = \{0, 1, \dots, n_k - 1\}$, for some $n_k \in \mathbb{N}$

Assume the **range** $\mathbb{R}^{\geq 0} = [0, +\infty)$

An (edge-valued) MxD is an acyclic directed edge-labeled graph where:

- The only **terminal** node is Ω and is at **level 0** $\Omega.lvl = 0$
- A **nonterminal** node p is at a **level** k , with $L \geq k \geq 1$ $p.lvl = k$
- A nonterminal node p at level k has $n_k \times n_k$ outgoing edges
- For $i_k, i'_k \in \mathcal{X}_k$, edge $p[i_k, i'_k]$ points to **child** $p[i_k, i'_k].child$, and has **value** $p[i_k, i'_k].val \geq 0$
- The level of the children is lower than that of p $p[i_k, i'_k].child.lvl < p.lvl$
- An edge $\langle \sigma, p \rangle$, with $p.lvl = k$ encodes the **function** $v_{\langle \sigma, p \rangle} : \mathcal{X}_{pot} \rightarrow \mathbb{Z}$ defined recursively by

$$v_{\langle \sigma, p \rangle}(x_L, x'_L, \dots, x_1, x'_1) = \begin{cases} \sigma & \text{if } k = 0 \\ \sigma \cdot v_{p[x_k, x'_k]}(x_L, x'_L, \dots, x_1, x'_1) & \text{if } k > 0 \end{cases}$$

For **canonical** MxDs, we first **normalize** each node p in one of two ways:

- $\max\{p[i_k, i'_k].val : i_k, i'_k \in \mathcal{X}_k\} = 1$, or
- $\min\{p[i_k, i'_k].val : i_k, i'_k \in \mathcal{X}_k, p[i_k, i'_k].val \neq 0\} = 1$

Then, the usual reduction requirements apply, there are no **duplicates**:

- If $p.lvl = q.lvl = k$ and $p[i_k] = q[i_k]$ for all $i_k \in \mathcal{X}_k$, then $p = q$

And, if the MxD is **quasi-reduced**, there is **no level skipping**:

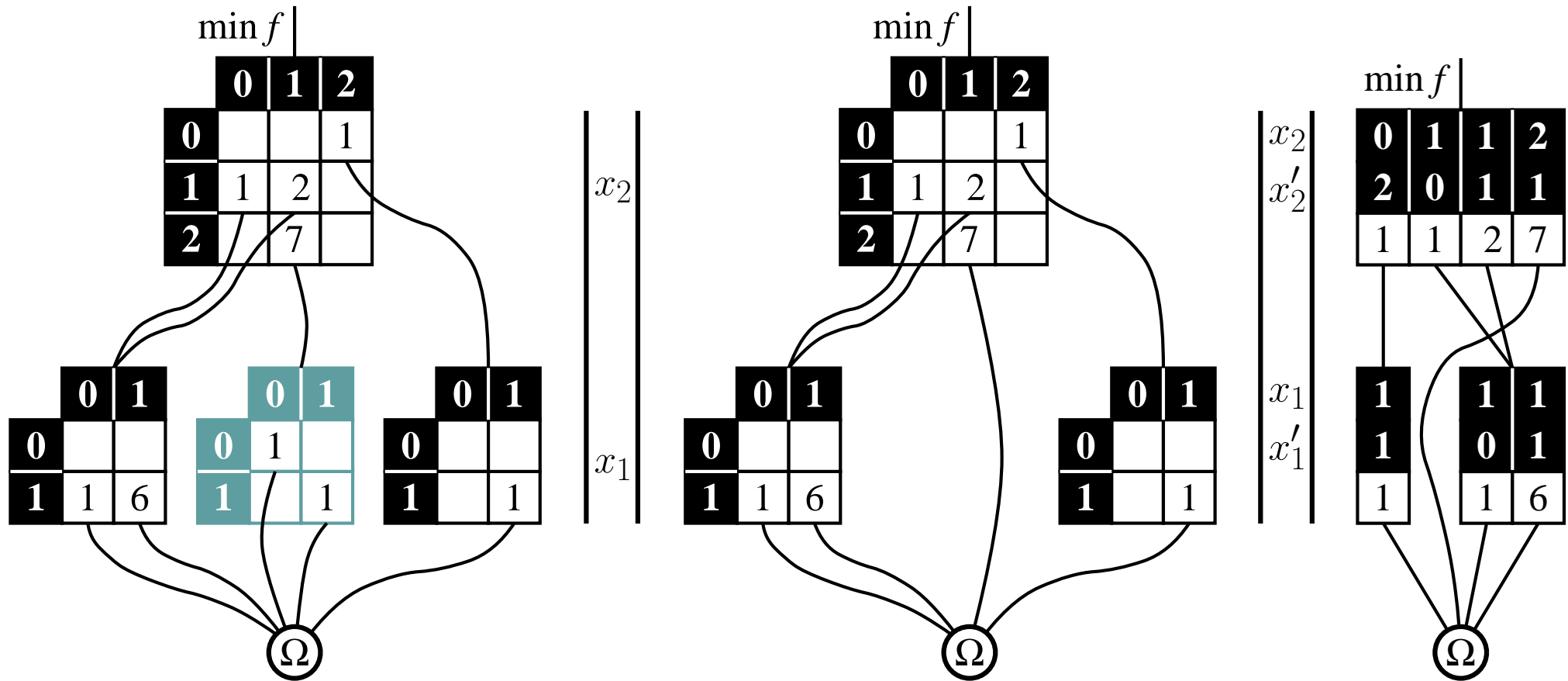
- The only **root** nodes with no incoming arcs are at level L , and have **root edge values** in \mathbb{Z}
- Each child $p[i_k, i'_k].child$ of a node p is at level $p.lvl - 1$

Or, if the MxD is **fully-reduced**, there is no **redundant node** p satisfying:

- $p[i_k, i'_k].child = q$ and $p[i_k, i'_k].val = 1$ for all $i_k, i'_k \in \mathcal{X}_k$

Or, if the MxD is **identity-reduced**, there are no **identity nodes** p satisfying:

- $p[i_k, i_k].child = q$ and $p[i_k, i_k].val = 1$ for all $i_k \in \mathcal{X}_k$
- $p[i_k, i'_k].val = 0$ for all $i_k \neq i'_k$



Approximate steady-state solution of continuous-time Markov chains (CTMCs)

- numerical fixpoint iteration on L CTMCs defined on the MDD for \mathcal{X}_{reach} and the MxD for \mathbf{R}

Computation of the invariants (**p-semiflows**) of a Petri net

- uses MDDs over **integer** \mathbb{Z} , not natural \mathbb{N} , domains

Reachability problems for **timed** models where activities have an integer duration

- uses MDDs, EV^+ MDDs

Application: can we use decision diagrams to solve **Rubik's cube**?

Bisimulation: partition the states according to the next-state function \mathcal{N} or the transition rate matrix \mathbf{R}

Sequence decision diagrams: to encode and manipulate variable-length sequences of symbols

General **ETDDs** definition integrating MTMDDs, EV^+ MDDs, EV^* MDDs, etc.

General saturation scheme for **arbitrary fixpoint computations** on structured domains