Data Representation and Efficient Solution:

A Decision Diagram Approach

Gianfranco Ciardo

University of California, Riverside

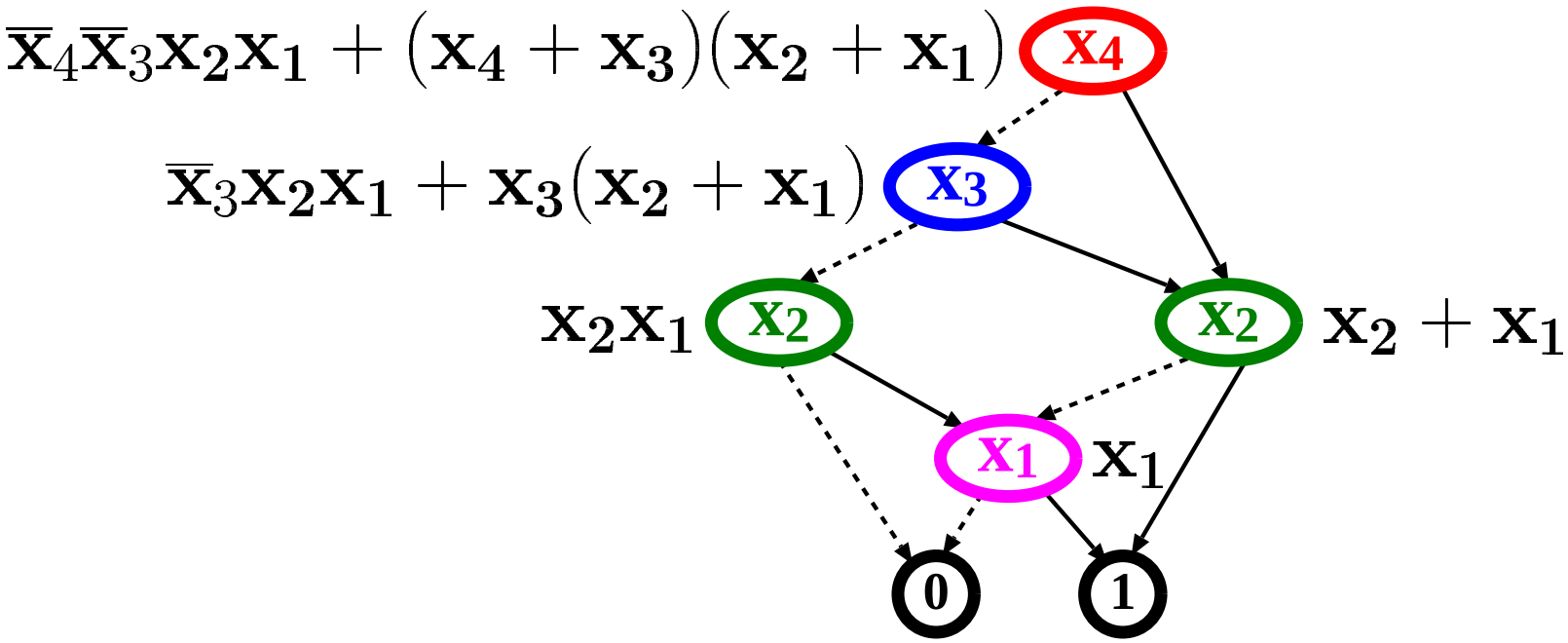
Decision diagrams: a static view

(Reduced ordered) binary decision diagrams (BDDs)

"Graph-based algorithms for boolean function manipulation"

Randy Bryant (Carnegie Mellon University)

IEEE Transactions on Computers, 1986



BDDs are a canonical representation of boolean functions $f : \{0,1\}^L \rightarrow \{0,1\}$

Ordered binary decision diagrams (BDDs)

A BDD is an acyclic directed edge-labeled graph where:

- The only terminal nodes can be $\mathbf{0}$ and $\mathbf{1}$, and are at level 0 $\quad \mathbf{0}.lvl = \mathbf{1}.lvl = 0$
- A nonterminal node p is at a level k, with $L \geq k \geq 1$ $\quad p.lvl = k$
- A nonterminal node p has two outgoing edges labelled 0 and 1, pointing to children $p[0]$ and $p[1]$
- The level of the children is lower than that of p; $\quad p[0].lvl < p.lvl, p[1].lvl < p.lvl$
- A node p at level k encodes the function $v_p : \mathbb{B}^L \rightarrow \mathbb{B}$ defined recursively by

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

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 $L \geq k \geq 1$

Canonical versions of BDDs

For canonical BDDs, we further require that

- There are no duplicates: if $p.lvl = q.lvl$ and $p[0] = q[0]$ and $p[1] = q[1]$, then $p = q$

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

- The only root nodes with no incoming arcs are at level L
- The children $p[0]$ and $p[1]$ of a node p are at level $p.lvl - 1$

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

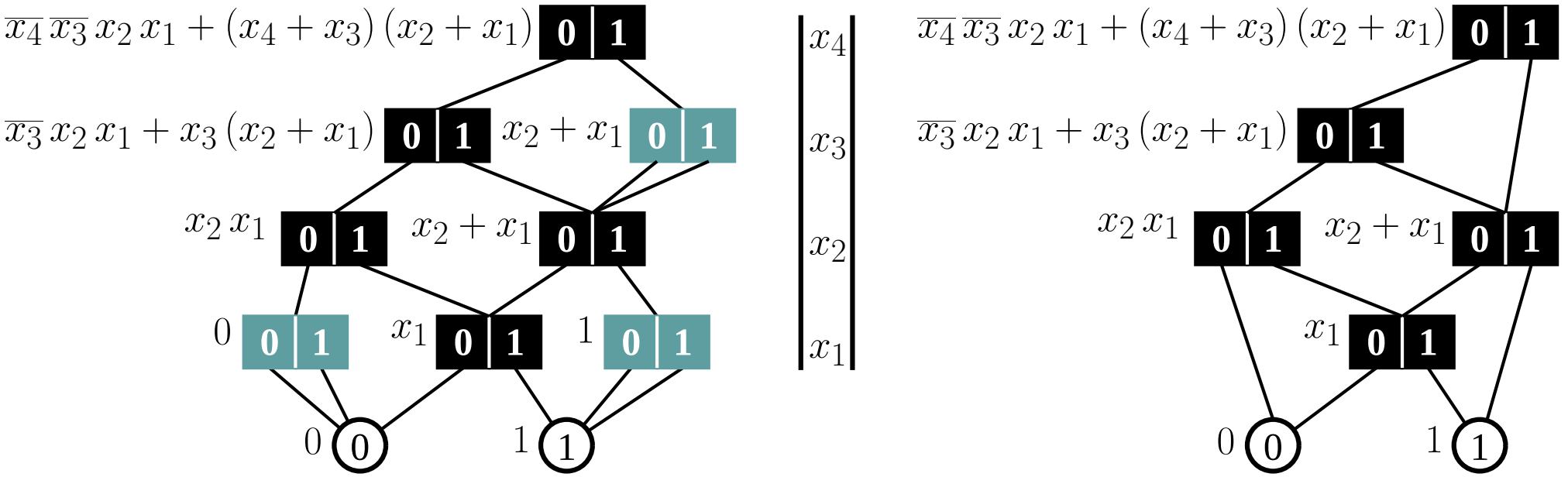
- There are no redundant nodes p satisfying $p[0] = p[1]$

Both versions are **canonical**, if functions f and g are encoded using BDDs:

- Satisfiability, $f \neq 0$, or equivalence, $f = g$ $O(1)$
- Conjunction, $f \wedge g$, disjunction, $f \vee g$, relational product: $O(|\mathcal{N}_f| \times |\mathcal{N}_g|)$, if fully-reduced
  $\sum_{L \geq k \geq 1} O(|\mathcal{N}_{f,k}| \times |\mathcal{N}_{g,k}|)$, if quasi-reduced

$\mathcal{N}_f$ = set of nodes in the BDD encoding f $\mathcal{N}_{f,k}$ = set of nodes at level k in the BDD encoding f

Quasi-reduced vs. fully reduced BDDs



Ordered multiway decision diagrams (MDDs)

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

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

- The only terminal nodes can be $\mathbf{0}$ and $\mathbf{1}$, and are at level 0 $\quad \mathbf{0}.lvl = \mathbf{1}.lvl = 0$
- A nonterminal node p is at a level k, with $L \geq k \geq 1$ $\quad p.lvl = k$
- 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; $\quad p[i_k].lvl < p.lvl$
- A node p at level k encodes the function $v_p : \widehat{\mathcal{X}} \to \mathbb{B}$ defined recursively by

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

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 $L \geq k \geq 1$

Canonical versions of MDDs

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 $p = q$

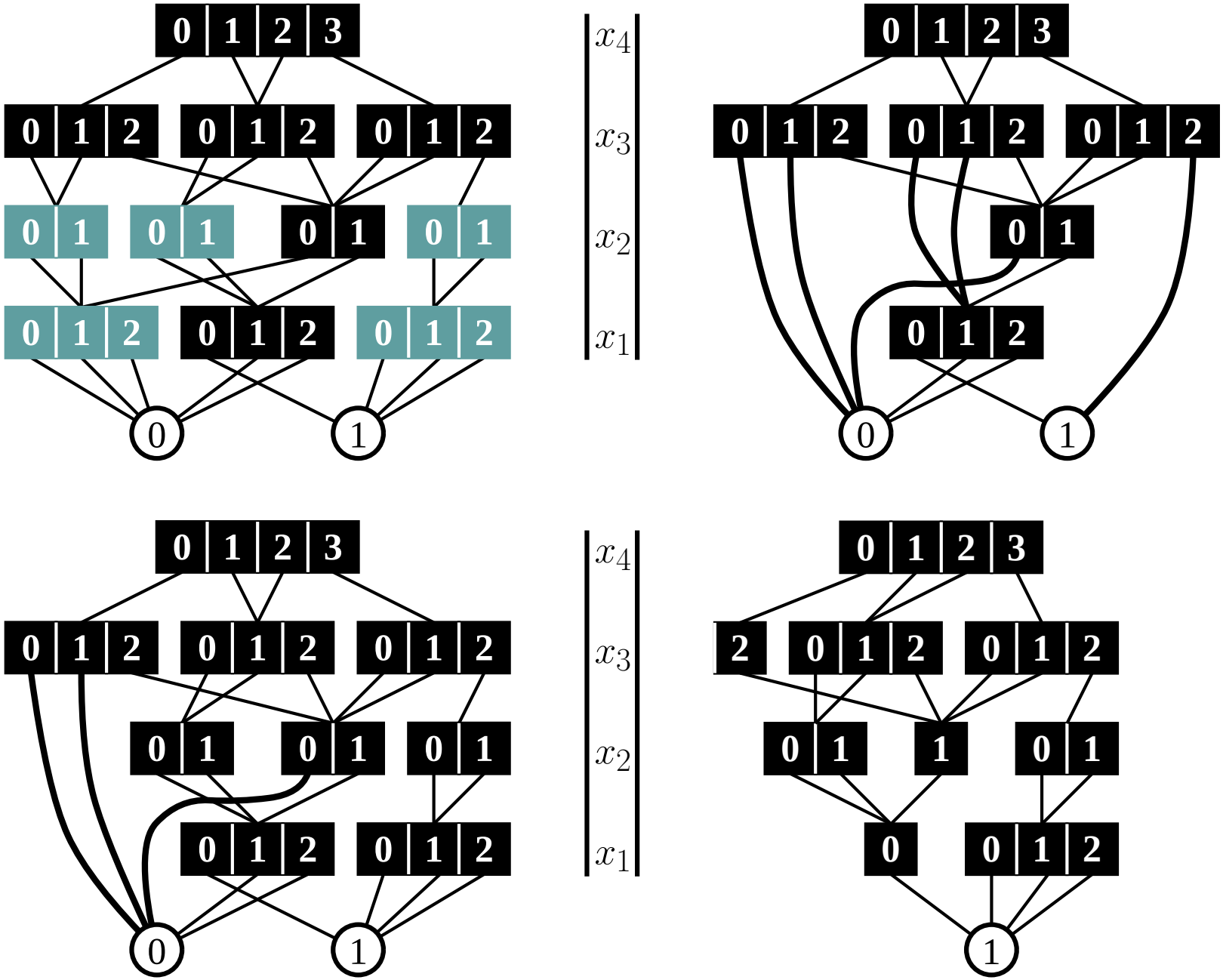
Then, if the MDD is quasi-reduced, there is no level skipping:

- The only root nodes with no incoming arcs are at level L
- Each child $p[i_k]$ 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 $p[i_k] = q$ for all $i_k \in \mathcal{X}_k$

Quasi-reduced vs. fully-reduced MDDs; full vs. sparse storage



Ordered multiterminal multiway decision diagrams (MTMDDs)

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

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

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 $v_p : \widehat{\mathcal{X}} \to \mathcal{X}_0$ defined recursively by

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

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 $L \geq k \geq 1$

Canonical versions of MTMDDs

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 $p = q$

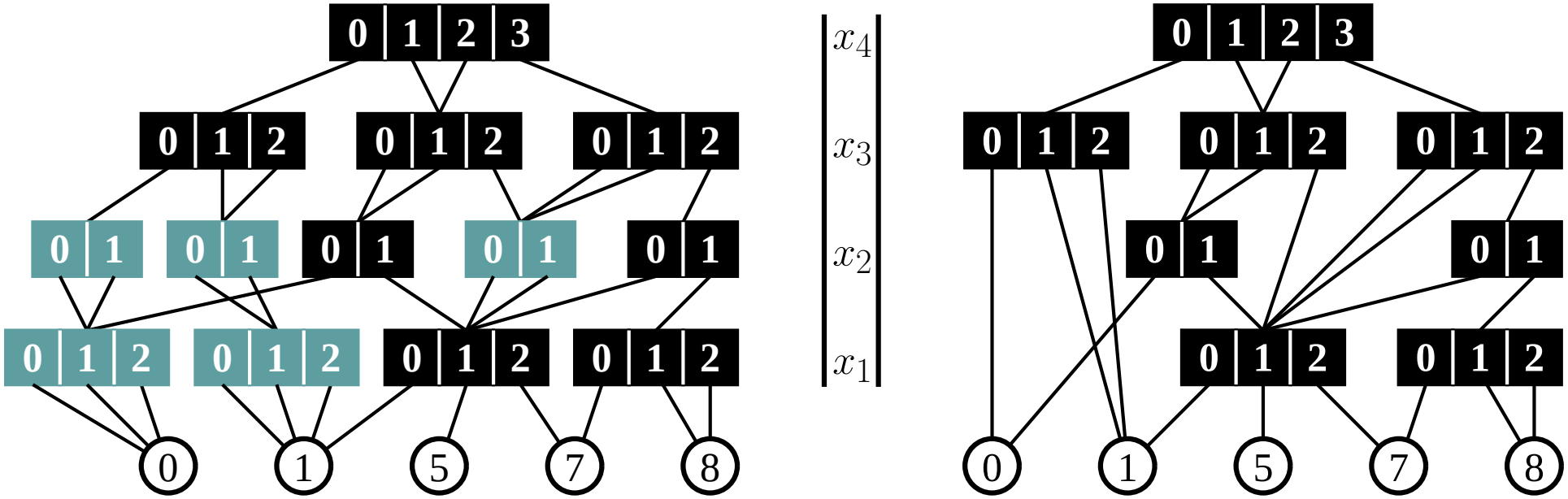
Then, if the MTMDD is quasi-reduced, there is no level skipping:

- 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 $i_k \in \mathcal{X}_k$

Quasi-reduced vs. fully reduced MTMDDs



Representing matrices with decision diagrams

A function $f : \widehat{\mathcal{X}} \to \mathcal{X}_0$ can be thought of as an $\mathcal{X}_0$-valued one-dimensional vector of size $|\widehat{\mathcal{X}}|$

We also need to store functions $\widehat{\mathcal{X}} \times \widehat{\mathcal{X}} \to \mathcal{X}_0$, or two-dimensional matrices

We can use a decision diagram with $2L$ levels:

- Unprimed x_k for the rows, or from, variables
- Primed x'_k for columns, or to variables
- Levels can be interleaved, $(x_L, x'_L, ..., x_1, x'_1)$, or non-interleaved, $(x_L, ..., x_1, x'_L, ..., x'_1)$

We can use a (terminal-valued) matrix diagram (MxD), analogous to a BDD, MDD, or MTMDD:

- A non-terminal node P at level k, for $L \geq k \geq 1$, has $n_k \times n_k$ edges
- $P[i_k, i'_k]$ points to the child corresponding to the choices $x_k = i_k$ and $x'_k = i'_k$

Identity patterns and identity-reduced decision diagrams

In the matrices that we need to encode, it is often the case that the entry is 0 if $x_k \neq x'_k$

An identity pattern in an interleaved $2L$-level MDD is

- a node p at level k
- with $p[i_k] = p'_{i_k}$
- such that $p'_{i_k}[i'_k] = 0$ for $i'_k \neq i_k$
- and $p'_{i_k} = q \neq 0$ only for $i'_k = i_k$

In an identity-reduced primed level k, we skip the nodes p'_{i_k}

An identity node in an MxD is

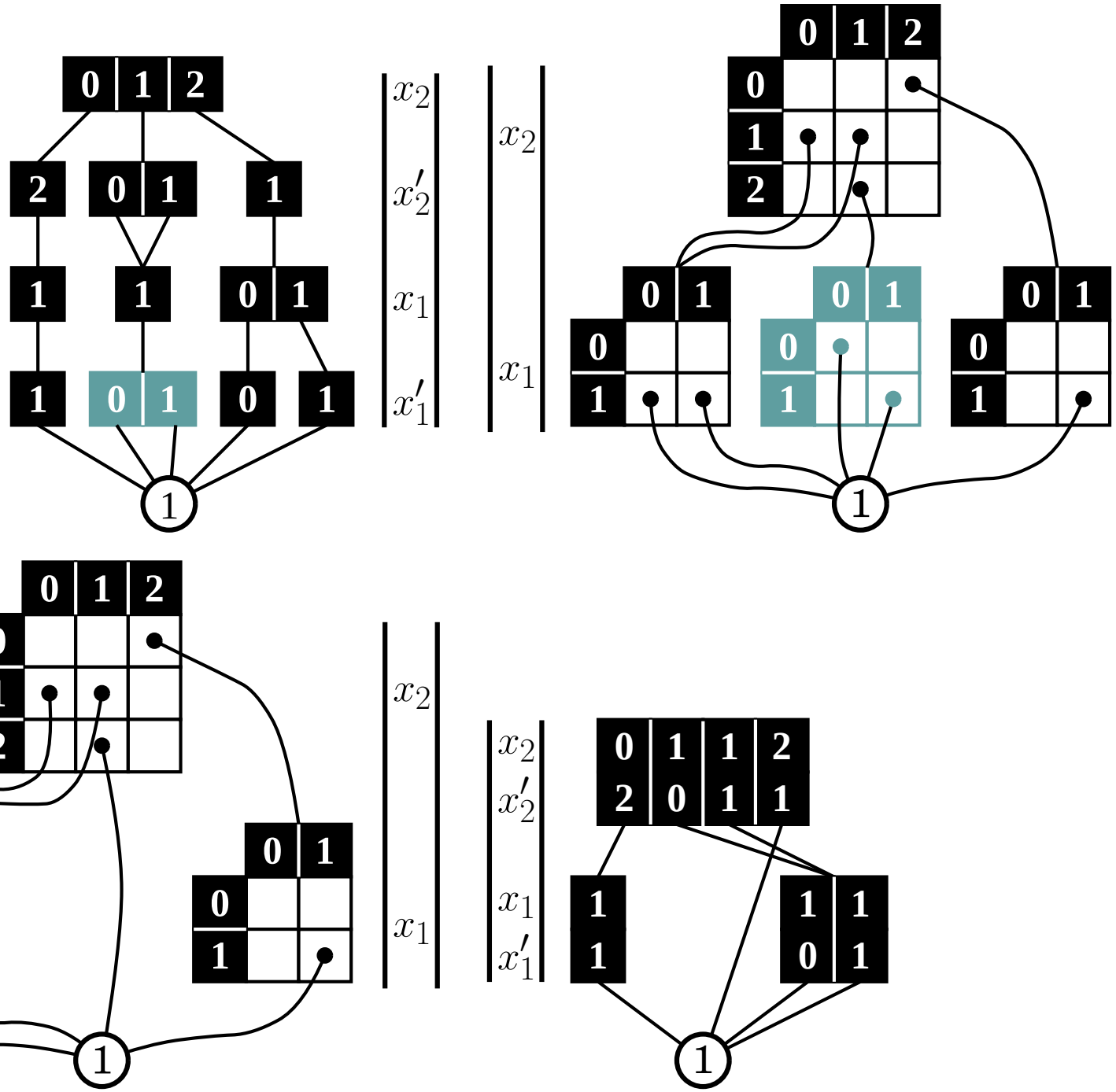
- a node P
- such that $P[i_k, i'_k] = 0$ for all $i_k, i'_k \in \mathcal{X}_k, i_k \neq i'_k$
- and $P[i_k, i_k] = q$ for all $i_k \in \mathcal{X}_k$

In an identity-reduced MxD, we skip these identity nodes

$2L$-level MDDs vs. MxDs: encoding a $(3 \cdot 2) \times (3 \cdot 2)$ matrix

$0 \equiv (x_2 = 0, x_1 = 0)$
$1 \equiv (x_2 = 0, x_1 = 1)$
$2 \equiv (x_2 = 1, x_1 = 0)$
$3 \equiv (x_2 = 1, x_1 = 1)$
$4 \equiv (x_2 = 2, x_1 = 0)$
$5 \equiv (x_2 = 2, x_1 = 1)$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	0
2	0	0	0	0	0	0
3	1	1	1	1	0	0
4	0	0	1	0	0	0
5	0	0	0	1	0	0



Ordered edge-valued multiway decision diagrams (EVMDDs)

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

Assume the range $\mathbb{Z}$ (can generalize to an arbitrary set)

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 (σ, p), with $p.lvl = k$ encodes the function $v_{(\sigma,p)} : \widehat{\mathcal{X}} \to \mathbb{Z}$ defined recursively by

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

Canonical versions of EVMDDs

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:

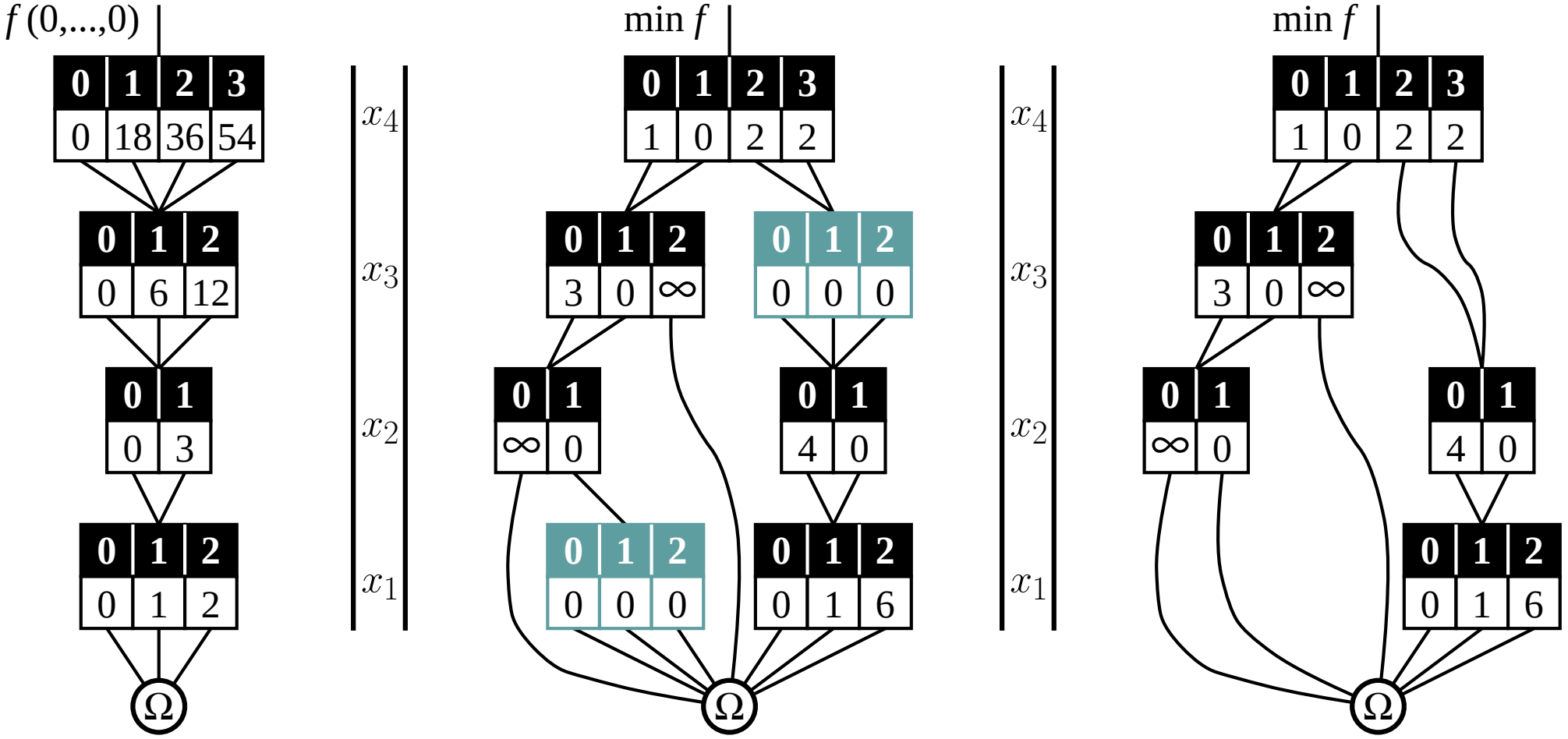
- There are no redundant nodes p satisfying $p[i_k].child = q$ and $p[i_k].val = 0$ for all $i_k \in \mathcal{X}_k$

EVMDDs, quasi-reduced EV$^+$MDDs, fully-reduced EV$^+$MDDs

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

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

The EV$^+$MDDs normalization allows to store partial functions $\widehat{\mathcal{X}} \to \mathbb{Z} \cup \{\infty\}$



Matrix diagrams (MxDs)

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

Assume the range $\mathbb{R}^{\geq 0} = [0, +\infty)$ (can generalize to an arbitrary set)

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 $P[i_k, jk].child$, 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 (σ, P), with $P.lvl = k$ encodes the function $v_{(\sigma,P)} : \widehat{\mathcal{X}} \to \mathbb{Z}$ defined recursively by

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

Canonical versions of MxDs

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 $p.lvl - 1$

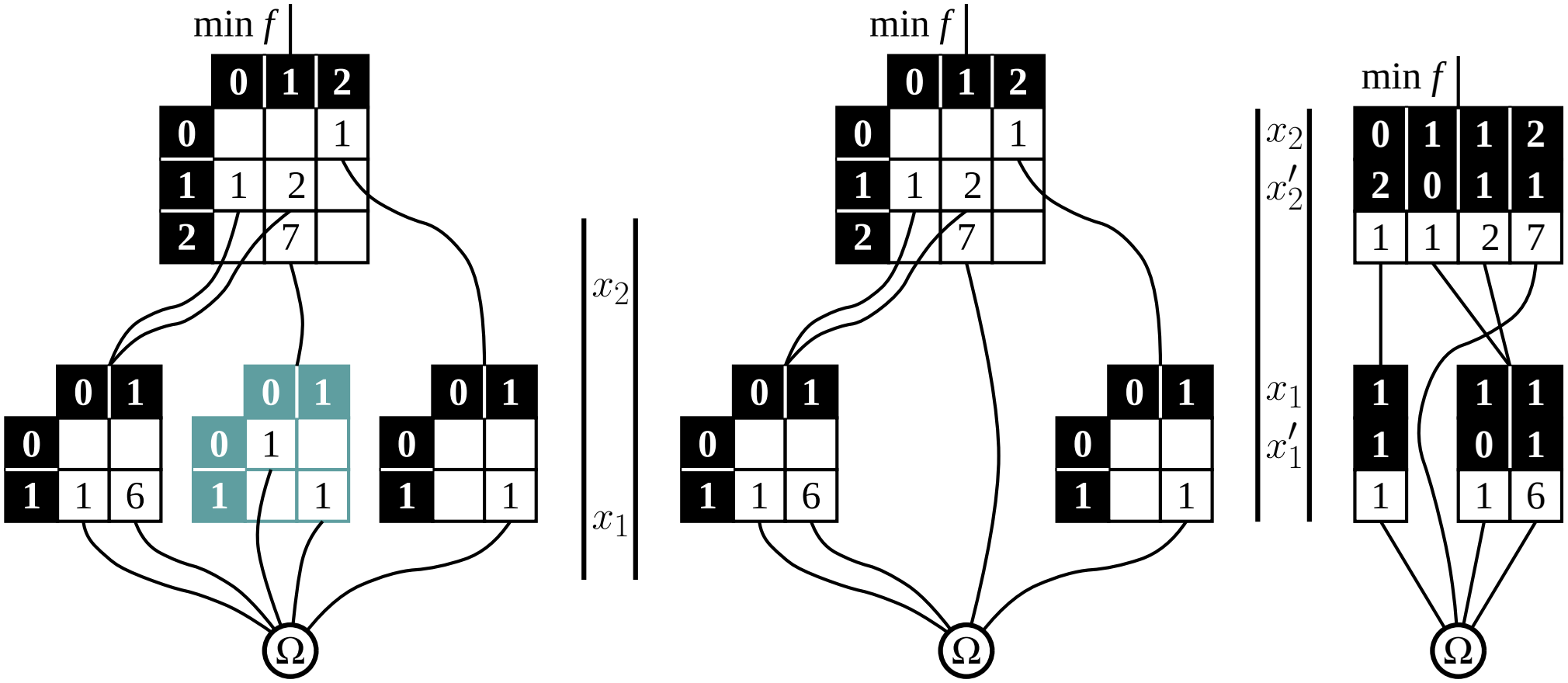
Or, if the MxD is fully-reduced, there is no redundant node P satisfying:

- $P[i_k, i'_k].child = Q$ 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 $i_k \neq i'_k$

Quasi-reduced vs. identity-reduced MxDs; sparse storage



Properties and applications

Properties of fully-reduced ordered BDDs

- Given a boolean expression, or a function, $f : \mathbb{B}^L \rightarrow \mathbb{B}$, there is a unique BDD encoding it (for a fixed variable order $x_L, \ldots, x_1$)
- Many functions have a very compact encoding as a BDD
- The constant functions 0 and 1 are represented by the nodes $\mathbf{0}$ and $\mathbf{1}$, respectively
- Given the BDD encoding of a boolean expression f: test whether $f \equiv 0$ or $f \equiv 1$ in $O(1)$ time
- Given the BDD encodings of boolean expressions f and g: test whether $f \equiv g$ in $O(1)$ time

- The variable ordering affects the size of the BDD, consider $\mathbf{x}_L \Leftrightarrow \mathbf{y}_L \wedge \cdots \wedge \mathbf{x}_1 \Leftrightarrow \mathbf{y}_1$
  - with the order $(x_L, y_L, \ldots, x_1, y_1)$ $O(L)$ nodes
  - with the order $(x_L, \ldots, x_1, y_L, \ldots, y_1)$ $O(2^L)$ nodes
- The BDD encoding of some functions is large (exponential) 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 BDD size is an NP-complete problem

MDDs to encode sets

An important application of BDDs and MDDs is to encode large sets to be manipulated symbolically

To encode a set $\mathcal{S} \subseteq \widehat{\mathcal{X}}$, we simply store its indicator function $\chi_{\mathcal{S}}$ in a decision diagram:

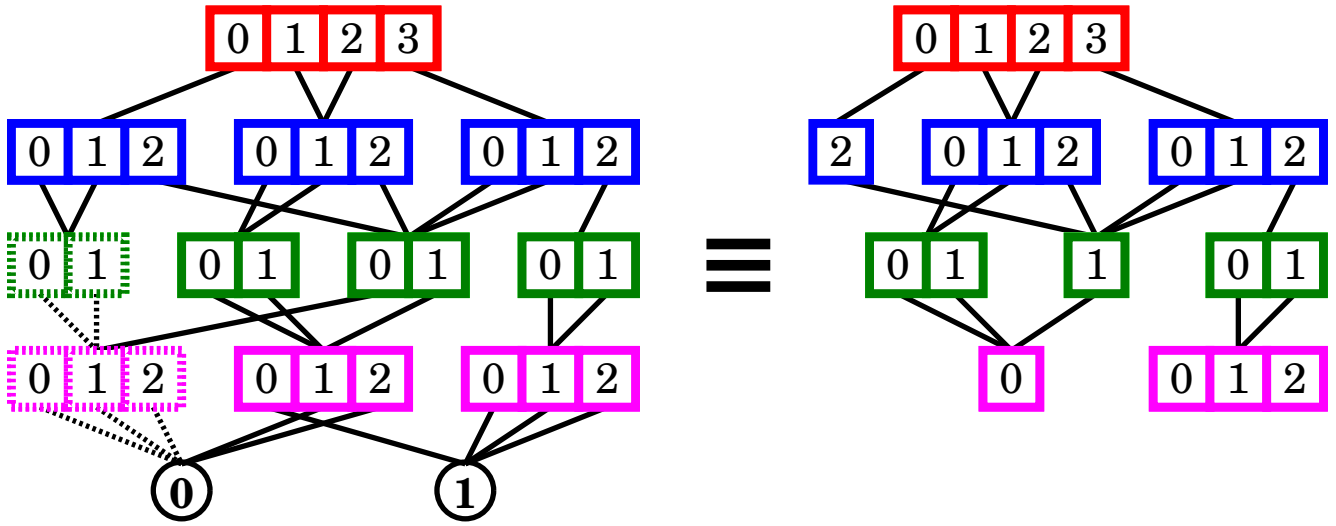
$$\chi_{\mathcal{S}}(i_L, ..., i_1) = 1 \Leftrightarrow (i_L, ..., i_1) \in \mathcal{S}$$

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

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

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

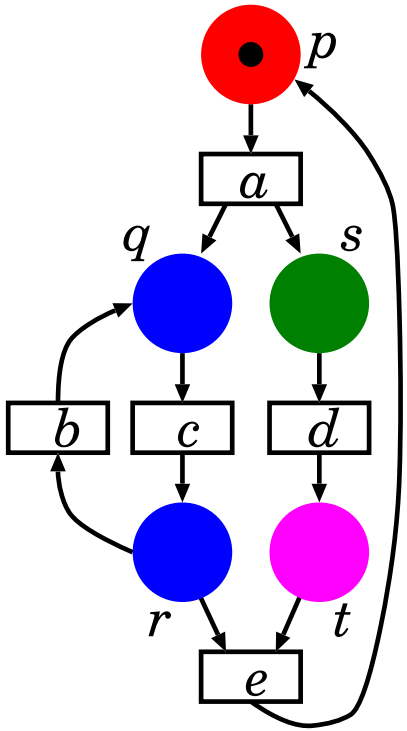
$\mathcal{X}_1 = \{0, 1, 2\}$

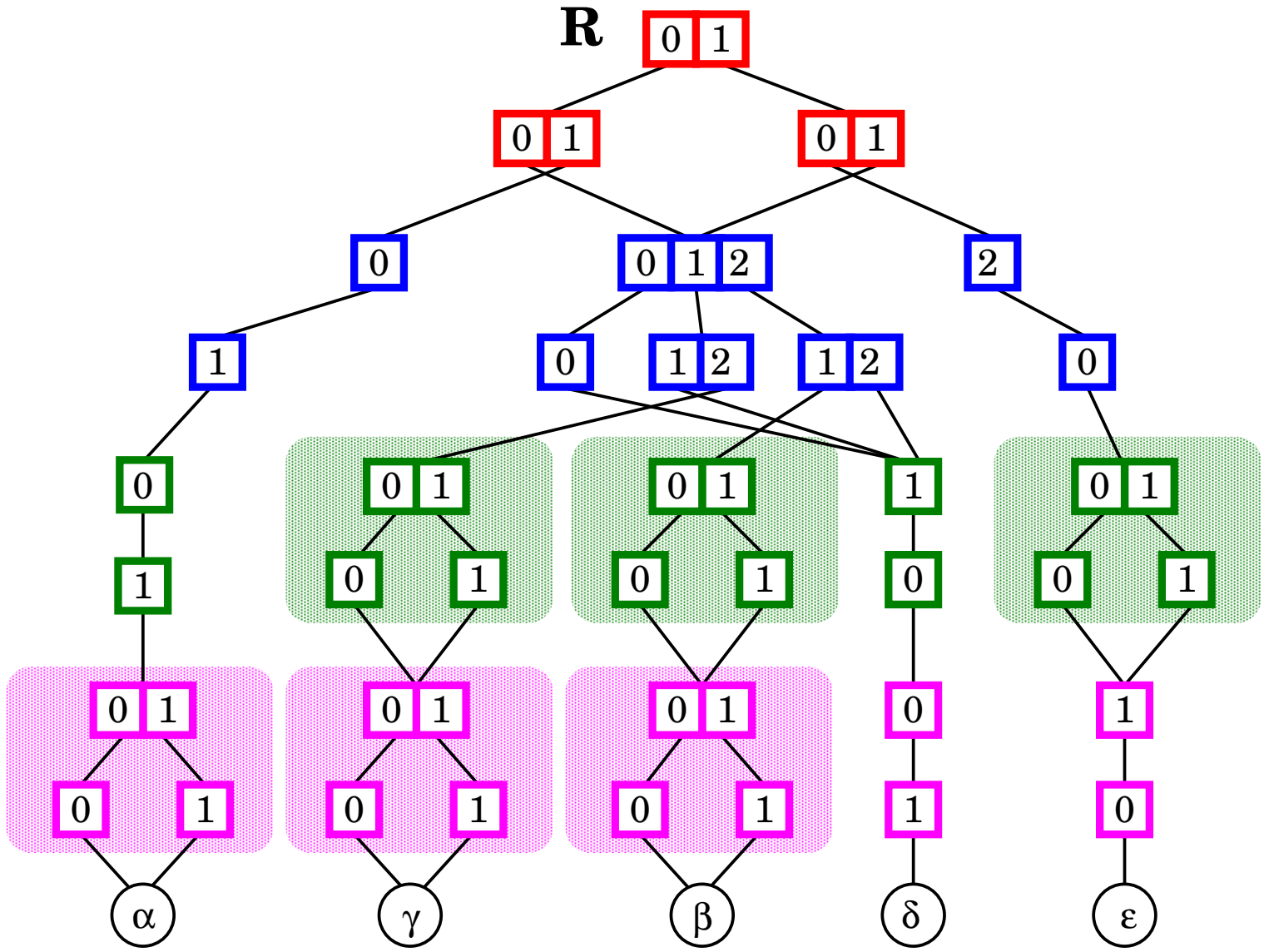


$$\mathcal{S} = \left\{ \begin{matrix} 0 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 0 & 0 & 1 & 1 & 2 & 0 & 0 & 1 & 1 & 2 & 0 & 1 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 1 & 2 \end{matrix} \right\}$$

An example of MTMDD: the transition rate matrix of an SPN

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



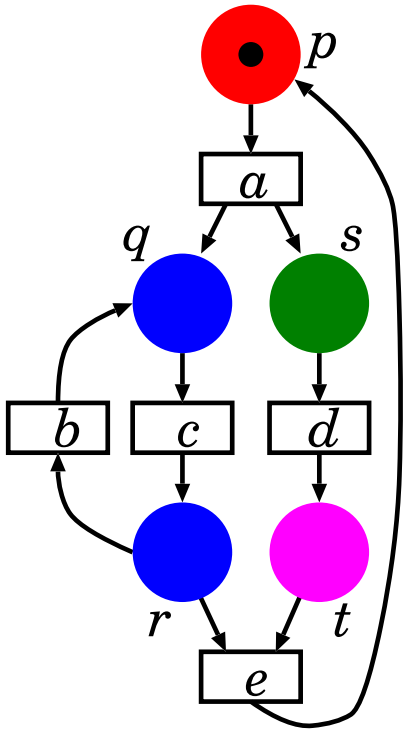


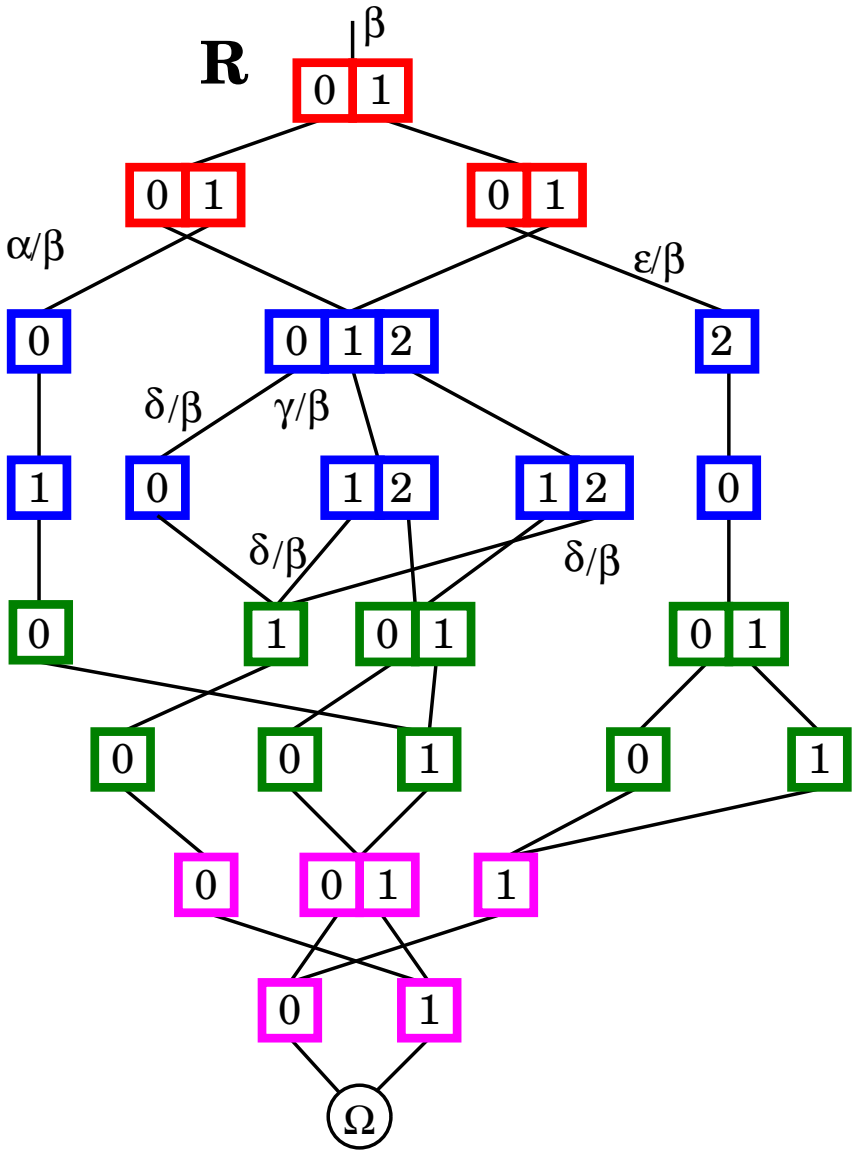
α = rate of a
β = rate of b
γ = rate of c
δ = rate of d
ϵ = rate of e

note the shaded identity patterns!!!

An example of EV*MDD: the transition rate matrix of an SPN

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





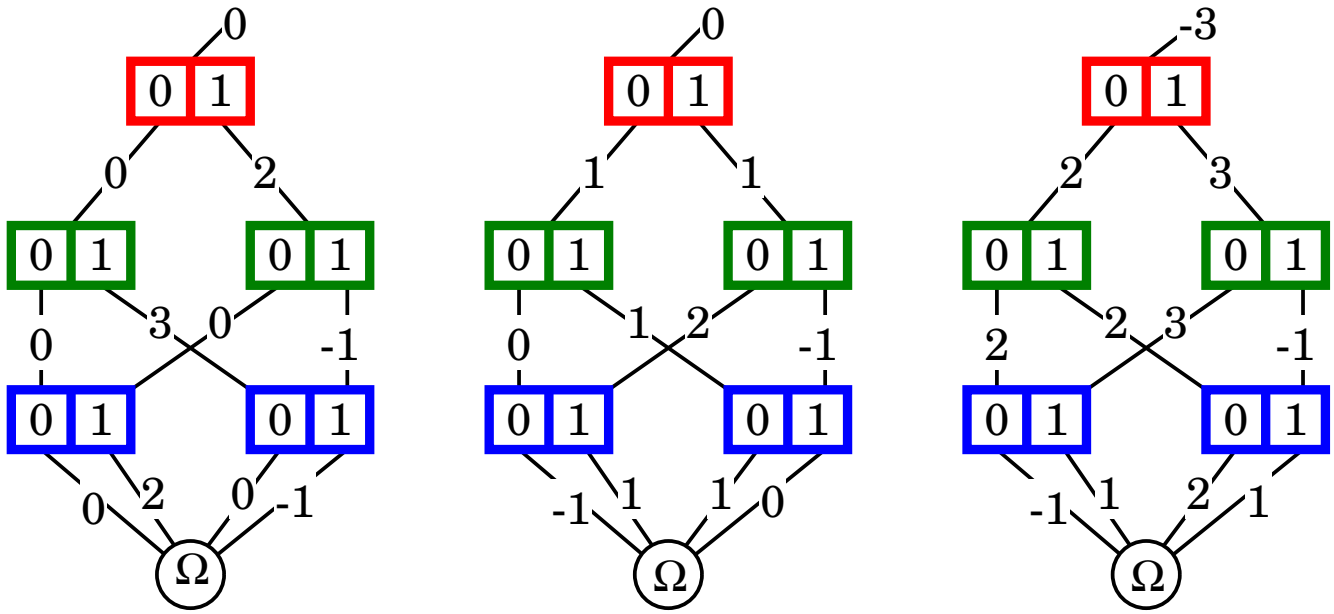
$\alpha =$ rate of a
$\beta =$ rate of b
$\gamma =$ rate of c
$\delta =$ rate of d
$\epsilon =$ rate of e

hidden identity patterns remain!!!

An example of EVBDDs

[Lai et al. 1992] defined edge-valued binary decision diagrams

$\mathbf{i}_3$	0 0 0 0 1 1 1 1
$\mathbf{i}_2$	0 0 1 1 0 0 1 1
$\mathbf{i}_1$	0 1 0 1 0 1 0 1
f	0 2 3 2 2 4 1 0



Canonicity: all nodes have a value 0 associated to the 0-arc (only the EVBDD on the left is canonical)

In canonical form, the root edge has value $f(0,\ldots,0)$

An example of EV$^+$MDDs

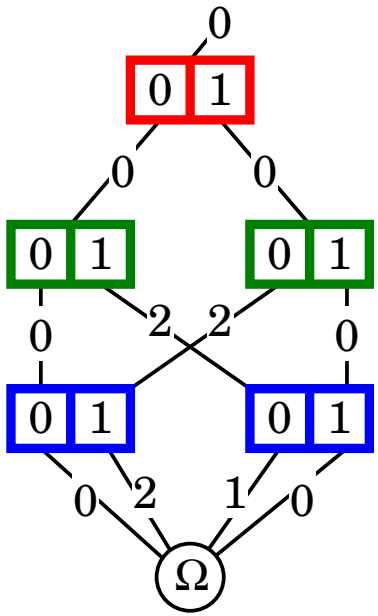
[CiaSim FMCAD'02] defined edge-valued positive multiway decision diagrams

From BDD to MDD: the usual extension

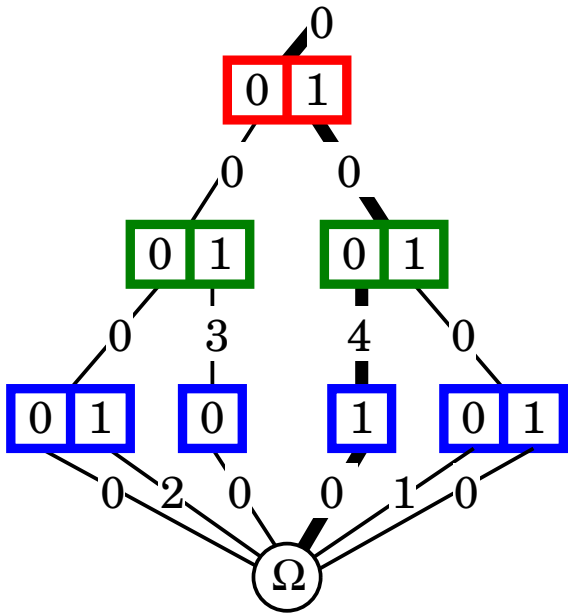
∞-edge values: can store partial arithmetic functions

Canonization rule different from that of EVBDDs: essential to encode partial arithmetic functions

$\mathbf{i}_3$	0 0 0 0 1 1 1 1
$\mathbf{i}_2$	0 0 1 1 0 0 1 1
$\mathbf{i}_1$	0 1 0 1 0 1 0 1
f	0 2 3 2 2 4 1 0



$\mathbf{i}_3$	0 0 0 0 1 1 1 1
$\mathbf{i}_2$	0 0 1 1 0 0 1 1
$\mathbf{i}_1$	0 1 0 1 0 1 0 1
f	0 2 3 ∞ ∞ 4 1 0



Canonicity: all edge values are non-negative and at least one is zero

In canonical form, the root edge has value $\min_{\mathbf{i}\in\widehat{\mathcal{X}}} f(\mathbf{i})$

$$f(1,0,0) = \infty \quad \text{but} \quad f(1,0,1) = 4$$

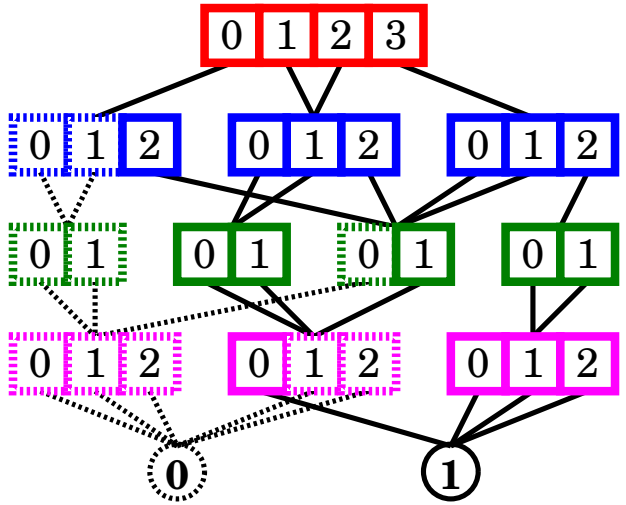
EV$^+$MDDs to store the lexicographic state indexing function ψ

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

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

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

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



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

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

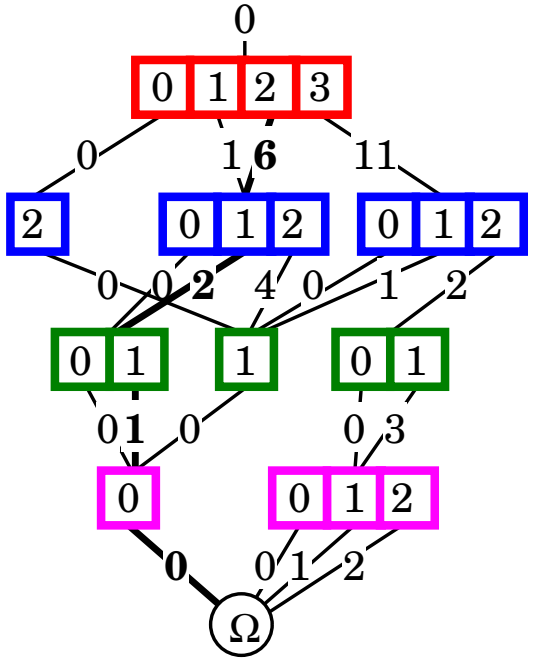
- Sum the values found on the corresponding path:

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

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

$$\psi(0, 2, 0, 0) = \mathbf{0} + \mathbf{0} + \infty = \infty$$

lexicographic, not discovery, order!!!



Decision diagrams: a dynamic view

The unique table and the operation cache

To ensure canonicity, thus greater efficiency, all operations use a Unique Table (a hash table):

- Search key: level $p.lvl$ and edges $p[0], ..., p[n_k - 1]$ of a node p Return value: a $node\_id$
- Alternative: one UT per level, no need to store $p.lvl$, but more fragmentation
- All (non-dead) nodes are referenced by the UT
- Collision must be without loss, multiple nodes with different $node\_id$ may have the same $hash\_val$

With the UT, we avoid duplicate nodes

To achieve polynomial complexity, all operations use an Operation Cache (a hash table):

- Search key: op_code and operands $node\_id_1$,$node\_id_2$,... Return value: $node\_id$
- Alternative: one OC per operation type, no need to store op_code, but more fragmentation
- Before computing $op\_code(node\_id_1, node\_id_2, ...)$, we search the OC
- If the search is successful, we avoid recomputing a result.
- Collision can be managed either without loss or with loss

With the OC, we visit every node combination instead of traveling every path

$Union$ (Or) and $Intersection$ (And) for fully-reduced BDDs

$bdd\ Union(bdd\ p, bdd\ q)$ is

local $bdd\ r$;

1 if $p = \mathbf{0}$ or $q = \mathbf{1}$ then return q;
2 if $q = \mathbf{0}$ or $p = \mathbf{1}$ then return p;
3 if $p = q$ then return p;
4 if $UnionCache$ contains entry $\langle \{p, q\} : r \rangle$ then return r;
5 if $p.lvl = q.lvl$ then
6 $\quad r \leftarrow UniqueTableInsert(p.lvl, Union(p[0], q[0]), Union(p[1], q[1]))$;
7 else if $a.lvl > b.lvl$ then
8 $\quad r \leftarrow UniqueTableInsert(p.lvl, Union(p[0], q), Union(p[1], q))$;
9 else since $a.lvl < b.lvl$ then
10 $\quad r \leftarrow UniqueTableInsert(q.lvl, Union(p, q[0]), Union(p, q[1]))$;
11 enter $\langle \{p, q\} : r \rangle$ in $UnionCache$;
12 return r;

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

$Union$: if $p = \mathbf{0}$ or $q = \mathbf{1}$ then return q;
if $q = \mathbf{0}$ or $p = \mathbf{1}$ then return p;

$Intersection$: if $p = \mathbf{1}$ or $q = \mathbf{0}$ then return q;
if $q = \mathbf{1}$ or $p = \mathbf{0}$ then return p;

complexity: $O(|\mathcal{N}_p| \times |\mathcal{N}_q|)$

$Union$ (Or) and $Intersection$ (And) for quasi-reduced MDDs

$mdd\ Union(lvl\ k, mdd\ p, mdd\ q)$ is

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

1 if $k = 0$ then return $p \vee q$;
2 if $p = q$ then return p;
3 if $UnionCache$ contains entry $\langle (k, \{p, q\}) = r \rangle$ then return r;
4 for $i = 0$ to $n_k - 1$ do
5 $\quad r_i \leftarrow Union(k-1, p[i], q[i])$;
6 end for
7 $r \leftarrow UniqueTableInsert(k, r_0, \ldots, r_{n_k-1})$;
8 enter $\langle (k, \{p, q\}) = r \rangle$ in $UnionCache$;
9 return r;

p and q are $\mathbf{0}$ or $\mathbf{1}$

$Intersection(k, p, q)$ differs from $Union(k, p, q)$ only in the terminal case:

$Union$: if $k = 0$ then return $p \vee q$;

$Intersection$: if $k = 0$ then return $p \wedge q$;

complexity: $O(\sum_{L \geq k \geq 1} |\mathcal{N}_{p,k}| \times |\mathcal{N}_{q,k}|)$

ITE and the $Apply$ operator for fully-reduced BDDs

The if-then-else, or ITE, ternary operator is defined as $ITE(f, g, h) = (f \wedge g) \vee (\neg f \wedge h)$

Let $f[c/x_k]$ be the function obtained from f by substituting variable x_k with the constant $c \in \mathbb{B}$

Then, $f = ITE(x_k, f[1/x_k], f[0/x_k])$ is the Shannon expansion of f with respect to variable x_k

For any binary boolean operator $\odot$: $ITE(x, u, v) \odot ITE(x, y, z) = ITE(x, u \odot y, v \odot z)$

This is the basis for the recursive BDD operator $Apply$

$bdd\ Apply(operator\ \odot, bdd\ p, bdd\ q)$ is

local $bdd\ \ r$;

1 if $p \in \{\mathbf{0}, \mathbf{1}\}$ and $q \in \{\mathbf{0}, \mathbf{1}\}$ then return $p \odot q$;
2 if $OperationCache$ contains entry $\langle \odot, p, q : r \rangle$ then return r;
3 if $p.lvl = q.lvl$ then
4 $\quad r \leftarrow UniqueTableInsert(p.lvl, Apply(\odot, p[0], q[0]), Apply(\odot, p[1], q[1]))$;
5 else if $p.lvl > q.lvl$ then
6 $\quad r \leftarrow UniqueTableInsert(p.lvl, Apply(\odot, p[0], q), Apply(\odot, p[1], q))$;
7 else since $p.lvl < q.lvl$ then
8 $\quad r \leftarrow UniqueTableInsert(q.lvl, Apply(\odot, p, q[0]), Apply(\odot, p, q[1]))$;
9 enter $\langle \odot, p, q : r \rangle$ in $OperationCache$;
10 return r;

Computing the relational product symbolically

Given an L-level BDD rooted at p^* encoding a set $\mathcal{X} \subseteq \widehat{\mathcal{X}}$ of states

Given a $2L$-level BDD rooted at P^*, encoding a relation $\mathcal{T}$ over $\widehat{\mathcal{X}}$

The call $RelationalProduct(p^*, P^*)$ returns the root r of the BDD encoding the set of states:

$$\mathcal{Y} = \{\mathbf{j} : \exists \mathbf{i} \in \mathcal{X} \ \wedge \ \exists (\mathbf{i}, \mathbf{j}) \in \mathcal{T}\}$$

$bdd\ RelationalProduct(bdd\ p, bdd\ P)$ is *quasi-reduced version*

local $bdd\ \ r, r_1, r_2$;

1 if $p = \mathbf{0}$ or $P = \mathbf{0}$ then return $\mathbf{0}$;
2 if $p = \mathbf{1}$ and $P = \mathbf{1}$ then return $\mathbf{1}$;
3 if $RelationalProductCache$ contains entry $\langle p, P : r \rangle$ then return r;
4 $r_0 \leftarrow Union(RelationalProduct(p[0], P[0][0]), RelationalProduct(p[1], P[1][0]))$;
5 $r_1 \leftarrow Union(RelationalProduct(p[0], P[0][1]), RelationalProduct(p[1], P[1][1]))$;
6 $r \leftarrow UniqueTableInsert(p.lvl, r_0, r_1)$;
7 enter $\langle p, P : r \rangle$ in $RelationalProductCache$;

The above algorithm assumes that:

- the order of the variables for $\mathcal{X}$ is $(x_L, ..., x_1)$
- the order of the variables for $\mathcal{T}$ is $(x_L, x'_L ..., x_1, x'_1)$

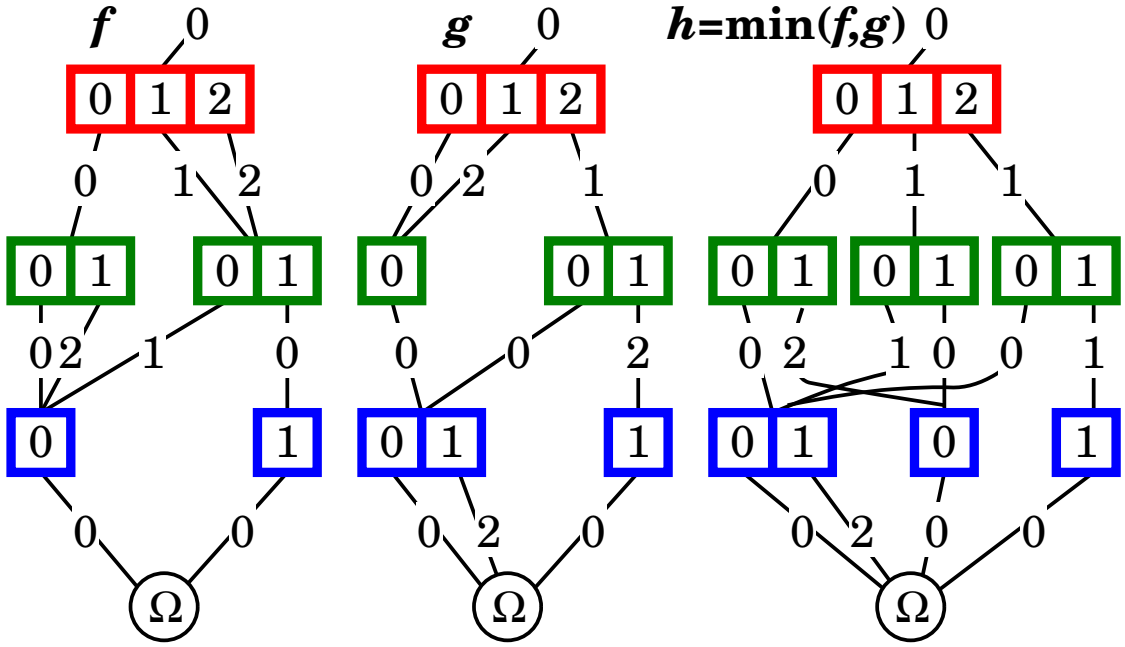
The Min operator for quasi-reduced EV$^+$MDDs

$edge\ Min(level\ k, edge\ (\alpha, p), edge\ (\beta, q))$ — *$edge$ is a pair $(int, node)$*

local $node$ p', q', r;
local int μ, α', β';
local $local$ i_k;
1 if $\alpha = \infty$ then return (β, q);
2 if $\beta = \infty$ then return (α, p);
3 $\mu \leftarrow \min(\alpha, \beta)$;
4 if $k = 0$ then return (μ, Ω); — *the only node at level 0 is Ω*
5 if $MinCache$ contains entry $\langle k, p, q, \alpha - \beta : \gamma, r \rangle$ then return $(\gamma + \mu, r)$;
6 $r \leftarrow NewNode(k)$; — *create new node at level k with edges set to (∞, Ω)*
7 for $i_k = 0$ to $n_k - 1$ do
8 $\quad p' \leftarrow p.child[i_k]$;
9 $\quad \alpha' \leftarrow \alpha - \mu + p.val[i_k]$;
10 $\quad q' \leftarrow q.child[i_k]$;
11 $\quad \beta' \leftarrow \beta - \mu + q.val[i_k]$;
12 $\quad r[i_k] \leftarrow Min(k-1, (\alpha', p'), (\beta', q'))$; — *continue downstream*
13 $UniqueTableInsert(k, r)$;
14 enter $\langle k, p, q, \alpha - \beta : \mu, r \rangle$ in $MinCache$;
15 return (μ, r);

An example of the Min operator for quasi-reduced EV$^+$MDDs

$\mathbf{i}_3$	0	0	0	0	1	1	1	1	2	2	2	2
$\mathbf{i}_2$	0	0	1	1	0	0	1	1	0	0	1	1
$\mathbf{i}_1$	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



Structured system analysis

Definition of structured discrete-state model

A structured discrete-state model is specified by

- a potential state space $\widehat{\mathcal{X}} = \mathcal{X}_L \times \cdots \times \mathcal{X}_1$
  - 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 $\{0, 1, \ldots, n_k - 1\}$ n_k might be unknown a priori
- a set of initial states $\mathcal{X}_{init} \subseteq \widehat{\mathcal{X}}$
  - often there is a single initial state $\mathbf{x}_{init}$
- a set of events $\mathcal{E}$ defining a disjunctively-partitioned next-state function or transition relation
  - $\mathcal{T}_\alpha : \widehat{\mathcal{X}} \to 2^{\widehat{\mathcal{X}}}$ $\mathbf{j} \in \mathcal{T}_\alpha(\mathbf{i})$ iff state $\mathbf{j}$ can be reached by firing event α in state $\mathbf{i}$
  - $\mathcal{T} : \widehat{\mathcal{X}} \to 2^{\widehat{\mathcal{X}}}$ $\mathcal{T}(\mathbf{i}) = \bigcup_{\alpha \in \mathcal{E}} \mathcal{T}_\alpha(\mathbf{i})$
  - naturally extended to sets of states $\mathcal{T}_\alpha(\mathcal{X}) = \bigcup_{\mathbf{i} \in \mathcal{X}} \mathcal{T}_\alpha(\mathbf{i})$ and $\mathcal{T}(\mathcal{X}) = \bigcup_{\mathbf{i} \in \mathcal{X}} \mathcal{T}(\mathbf{i})$
  - α is enabled in $\mathbf{i}$ iff $\mathcal{T}_\alpha(\mathbf{i}) \neq \emptyset$, otherwise it is disabled
  - $\mathbf{i}$ is absorbing, or dead, if $\mathcal{T}(\mathbf{i}) = \emptyset$

Using BDDs to build the state space $\mathcal{X}_{reach}$

L-level BDD encodes a set of states $\mathcal{S}$ as a subset of the potential state space $\widehat{\mathcal{X}} = \{0,1\}^L$

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

$2L$-level BDD encodes the transition relation $\mathcal{T} \subseteq \widehat{\mathcal{X}} \times \widehat{\mathcal{X}}$

$(\mathbf{i}, \mathbf{j}) \equiv (i_L, j_L, ..., i_1, j_1) \in \mathcal{T} \Leftrightarrow$ the system can go from $\mathbf{i}$ to $\mathbf{j}$ in one step

We can also think of it as the next-state function $\mathcal{T} : \widehat{\mathcal{X}} \rightarrow 2^{\widehat{\mathcal{X}}}$

$\mathbf{j} \in \mathcal{T}(\mathbf{i}) \Leftrightarrow$ the system can go from $\mathbf{i}$ to $\mathbf{j}$ in one step

Standard method

$ExploreBdd(\mathcal{X}_{init}, \mathcal{T})$ is

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

Alternative All method

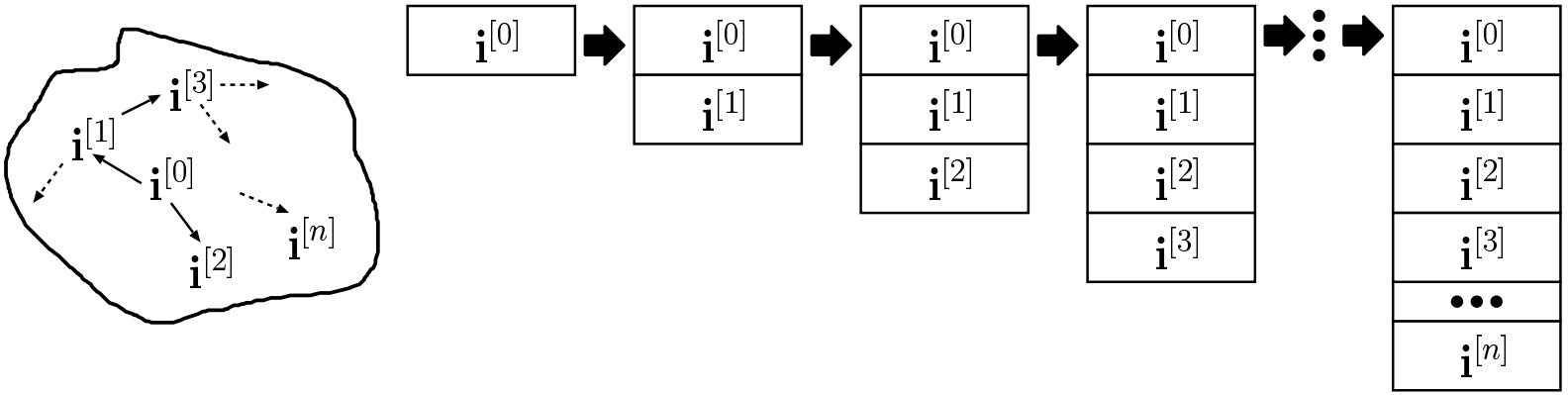
$AllExploreBdd(\mathcal{X}_{init}, \mathcal{T})$ is

1 $\mathcal{S} \leftarrow \mathcal{X}_{init}$;
2 repeat
3 $\quad \mathcal{O} \leftarrow \mathcal{S}$; *old states*
4 $\quad \mathcal{S} \leftarrow \mathcal{O} \cup \mathcal{T}(\mathcal{O})$; *new states*
5 until $\mathcal{O} = \mathcal{S}$;
6 return $\mathcal{S}$;

Explicit vs. symbolic state space generation

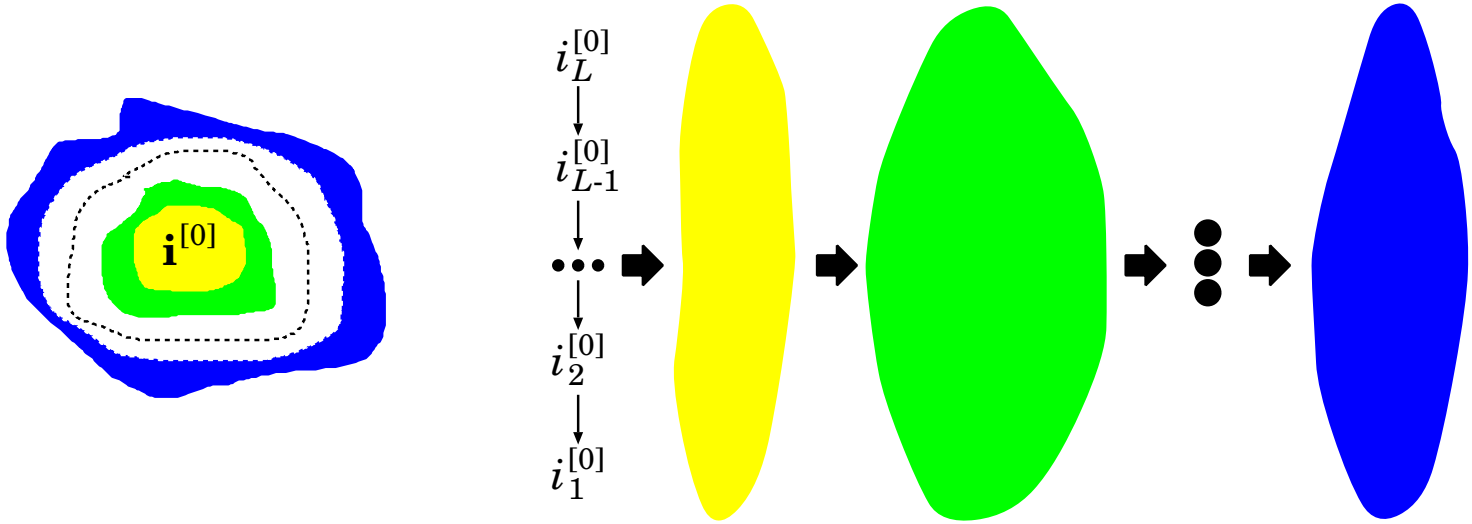
Explicit generation of the state space $\mathcal{X}_{reach}$ adds one state at a time

- memory O(states), increases linearly, peaks at the end

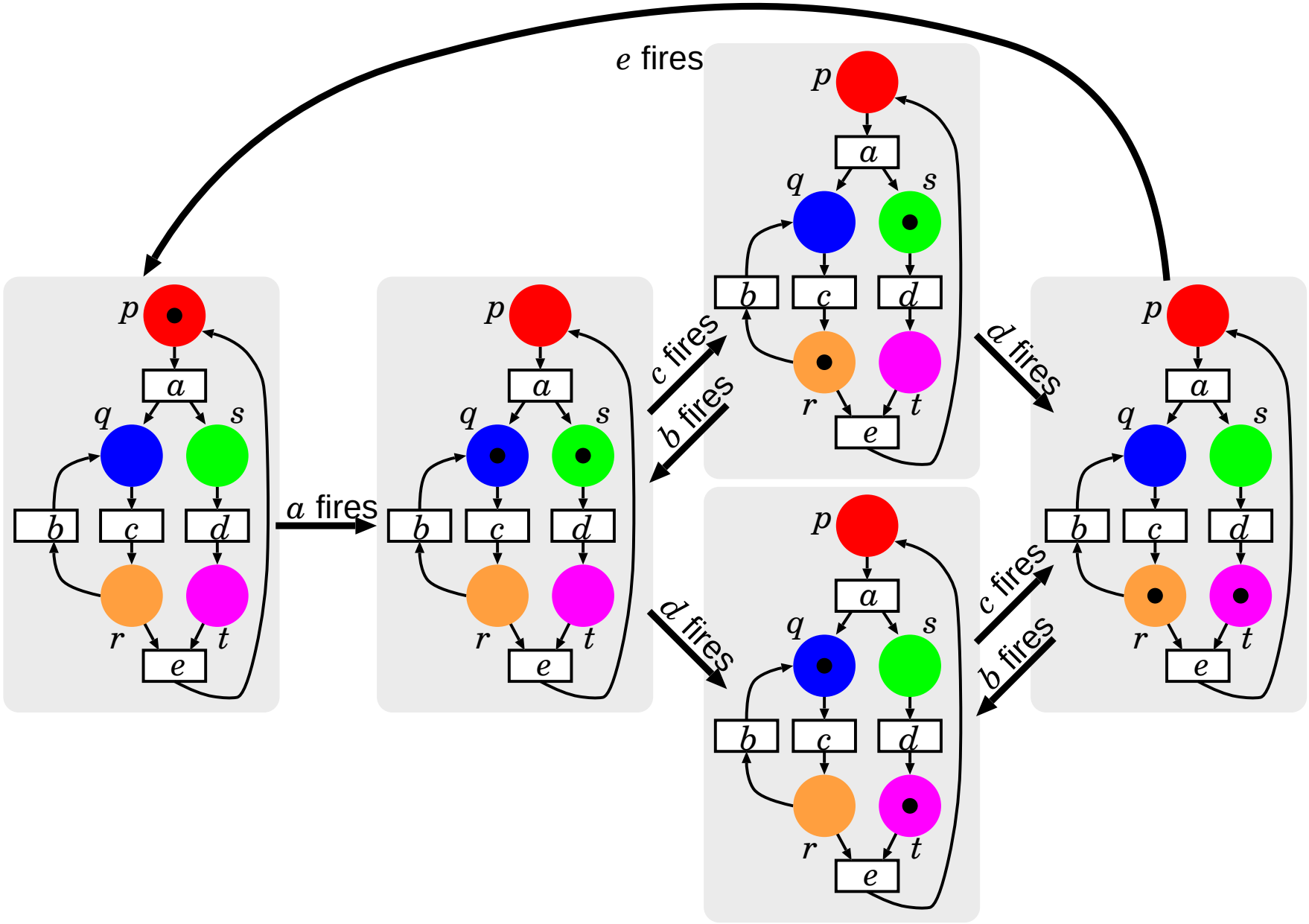


Symbolic generation of the state space $\mathcal{X}_{reach}$ with decision diagrams adds sets of states instead

- memory O(decision diagram nodes), grows and shrinks, usually peaks well before the end



Petri nets and their state space $\mathcal{X}_{reach}$: finite case



If the initial state is $\mathbf{x}_{init} = (N, 0, 0, 0, 0)$, $\mathcal{X}_{reach}$ contains $\dfrac{(N+1)(N+2)(2N+3)}{6}$ states

Self-modifying Petri nets with inhibitor arcs, guards, priorities

A self-modifying Petri net with inhibitor arcs, guards, and priorities is a tuple

$$(\mathcal{P}, \mathcal{E}, \mathbf{D}^-, \mathbf{D}^+, \mathbf{D}^\circ, G, \succ, \mathbf{x}_{init})$$

- $\mathcal{P}$ and $\mathcal{E}$ — places and events
- $\mathbf{D}^-, \mathbf{D}^+ : \mathcal{E} \times \mathcal{P} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N}$ — state-dependent input, output arc cardinalities
- $\mathbf{D}^\circ : \mathcal{E} \times \mathcal{P} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N} \cup \{\infty\}$ — state-dependent inhibitor arc cardinalities
- $G : \mathcal{E} \times \mathbb{N}^{|\mathcal{P}|} \to \{true, false\}$ — state-dependent guards
- $\succ \subset \mathcal{E} \times \mathcal{E}$ — acyclic (preselection) priority relation
- $\mathbf{x}_{init} : \mathbb{N}^{|\mathcal{P}|}$ — initial state

Event α is enabled in a state $\mathbf{i} \in \mathbb{N}^{|\mathcal{P}|}$, written $\alpha \in \mathcal{E}(\mathbf{i})$, iff

$$\forall p \in \mathcal{P},\ \mathbf{D}^-_{\alpha,p}(\mathbf{i}) \leq i_p \ \wedge\ \mathbf{D}^\circ_{\alpha,p}(\mathbf{i}) > i_p \ \wedge G_\alpha(\mathbf{i}) \ \wedge\ \forall \beta \in \mathcal{E},\ \beta \succ \alpha \ \Rightarrow\ \beta \notin \mathcal{E}(\mathbf{i})$$

If $\mathbf{i} \xrightarrow{\alpha} \mathbf{j}$, the new state $\mathbf{j}$ satisfies $\forall p \in \mathcal{P},\ \ j_p = i_p - \mathbf{D}^-_{\alpha,p}(\mathbf{i}) + \mathbf{D}^+_{\alpha,p}(\mathbf{i})$ (deterministic effect)

Symbolic state-space generation of safe Petri nets [Pastor94]

We can store

- any set of markings $\mathcal{X} \subseteq \widehat{\mathcal{X}} = \{0,1\}^{|\mathcal{P}|}$ of a safe PN with a $|\mathcal{P}|$-level BDD
- any relation over $\widehat{\mathcal{X}}$, or function $\widehat{\mathcal{X}} \to 2^{\widehat{\mathcal{X}}}$, such as $\mathcal{T}$, with a $2|\mathcal{P}|$-level BDD

We can encode $\mathcal{T}$ using $4|\mathcal{E}|$ boolean functions, each corrresponding to a very simple BDD

- $APM_\alpha = \prod_{p:\mathbf{F}^-p,\alpha=1}(x_p = 1)$ (all predecessor places of α are marked)
- $NPM_\alpha = \prod_{p:\mathbf{F}^-p,\alpha=1}(x_p = 0)$ (no predecessor place of α is marked)
- $ASM_\alpha = \prod_{p:\mathbf{F}^+p,\alpha=1}(x_p = 1)$ (all successor places of α are marked)
- $NSM_\alpha = \prod_{p:\mathbf{F}^+p,\alpha=1}(x_p = 0)$ (no successor place of α is marked)

Symbolic state-space generation of safe Petri nets (cont.)

The topological image computation for a transition α on a set of states $\mathcal{U}$ can be expressed as

$$\mathcal{T}_\alpha(\mathcal{U}) = (((\mathcal{U} \div APM_\alpha) \cdot NPM_\alpha) \div NSM_\alpha) \cdot ASM_\alpha$$

where "$\div$" indicates the cofactor operator and "$\cdot$" indicates boolean conjunction

Given

- a boolean function f over $(x_L, \ldots, x_1)$
- a literal $x_k = i_k$, with $L \geq k \geq 1$ and $i_k \in \mathbb{B}$

the cofactor $f \div (x_k = i_k)$ is defined as

- $f(x_L, \ldots, x_{k+1}, i_k, x_{k-1}, \ldots, x_1)$

The extension to multiple literals, $f \div (x_{k_c} = i_{k_c}, \ldots, x_{k_1} = i_{k_1})$, is recursively defined as

- $f(x_L, \ldots, x_{k_c+1}, i_{k_c}, x_{k_c-1}, \ldots, x_1) \div (x_{k_{c-1}} = i_{k_{c-1}}, \ldots, x_{k_1} = i_{k_1})$

Thus, $\mathcal{T}$ is stored in a disjunctively partition form as $\mathcal{T} = \bigcup_{\alpha \in \mathcal{E}} \mathcal{T}_\alpha$

Chaining [Roig95]

For a Petri net where $\mathcal{T}$ is stored in a disjunctively partitioned form, the effect of

$$\mathcal{X} \leftarrow \mathcal{T}(\mathcal{U});$$
$$\mathcal{U} \leftarrow \mathcal{X} \setminus \mathcal{S};$$

is exactly achieved with the statements

$$\mathcal{X} \leftarrow \emptyset;$$
for each $\alpha \in \mathcal{E}$ do
$$\quad \mathcal{X} \leftarrow \mathcal{X} \cup \mathcal{T}_\alpha(\mathcal{U});$$
$$\mathcal{U} \leftarrow \mathcal{X} \setminus \mathcal{S};$$

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

for each $\alpha \in \mathcal{E}$ do
$$\quad \mathcal{U} \leftarrow \mathcal{U} \cup \mathcal{T}_\alpha(\mathcal{U});$$
$$\quad \mathcal{U} \leftarrow \mathcal{U} \setminus \mathcal{S};$$

Symbolic $SsGen$: breadth-first vs. chaining, new vs. all states

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

```
1  S ← X_init;
2  U ← X_init;
3  repeat
4      X ← ∅;
5      for each α ∈ E do
6          X ← X ∪ T_α(U);
7      U ← X \ S;
8      S ← S ∪ U;
9  until U = ∅;
10 return S;
```

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

```
1  S ← X_init;
2  U ← X_init;
3  repeat
4      for each α ∈ E do
5          U ← U ∪ T_α(U);
6          U ← U \ S;
7      S ← S ∪ U;
8  until U = ∅;
9  return S;
```

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

```
1  S ← X_init;
2  repeat
3      O ← S;
4      X ← ∅;
5      for each α ∈ E do
6          X ← X ∪ T_α(O);
7      S ← O ∪ X;
8  until O = S;
9  return S;
```

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

```
1  S ← X_init;
2  repeat
3      O ← S;
4      for each α ∈ E do
5          S ← S ∪ T_α(S);
6  until O = S;
7  return S;
```

Comparing the four approaches

N	$\lvert\mathcal{X}_{reach}\rvert$	Time (sec) Bf	$AllBf$	Ch	$AllCh$	Memory (MB) Bf	$AllBf$	Ch	$AllCh$	final
Dining Philosophers: $L=N/2$, $\lvert\mathcal{X}_k\rvert=34$ for all k										
50	2.2×10^{31}	37.6	36.8	1.3	1.3	146.8	131.6	2.2	2.2	0.0
100	5.0×10^{62}	644.1	630.4	5.4	5.3	>999.9	>999.9	8.9	8.9	0.0
1000	9.2×10^{626}	—	—	895.4	915.5	—	—	895.2	895.0	0.3
Slotted Ring Network: $L=N$, $\lvert\mathcal{X}_k\rvert=15$ for all k										
5	5.3×10^{4}	0.2	0.3	0.1	0.1	0.8	1.1	0.3	0.2	0.0
10	8.3×10^{9}	21.5	24.1	2.1	1.2	39.0	45.0	5.7	3.3	0.0
15	1.5×10^{15}	745.4	771.5	18.5	8.9	344.3	375.4	35.1	20.2	0.0
Round Robin Mutual Exclusion: $L=N+1$, $\lvert\mathcal{X}_k\rvert=10$ for all k except $\lvert\mathcal{X}_1\rvert=N+1$										
10	2.3×10^{4}	0.2	0.3	0.1	0.1	0.6	1.2	0.1	0.1	0.0
20	4.7×10^{7}	2.7	4.4	0.3	0.3	5.9	12.8	0.5	0.5	0.0
50	1.3×10^{17}	263.2	427.6	2.9	2.8	126.7	257.7	4.3	3.8	0.1
FMS: $L=19$, $\lvert\mathcal{X}_k\rvert=N+1$ for all k except $\lvert\mathcal{X}_{17}\rvert=4$, $\lvert\mathcal{X}_{12}\rvert=3$, $\lvert\mathcal{X}_7\rvert=2$										
5	2.9×10^{6}	0.7	0.7	0.1	0.1	2.6	2.2	0.4	0.2	0.0
10	2.5×10^{9}	7.0	5.8	0.5	0.3	18.2	14.7	2.3	1.3	0.0
25	8.5×10^{13}	677.2	437.9	12.9	5.1	319.7	245.3	42.7	21.2	0.1

CTL: computation tree logic

Given a Kripke structure $(\widehat{\mathcal{X}}, \mathcal{X}_{init}, \mathcal{T}, \mathcal{A}, \mathcal{L})$

CTL has state formulas and path formulas

- State formulas:
  - if $a \in \mathcal{A}$, a is a state formula (a is an atomic proposition, true or false in each state)
  - if p and p' are state formulas, $\neg p$, $p \vee p'$, $p \wedge p'$ are state formulas
  - if q is a path formula, Eq, Aq are state formulas
- Path formulas:
  - if p and p' are state formulas, Xp, Fp, Gp, pUp', pRp' are path formulas
  - Note: unlike CTL*, a state formula is **not** also a path formula

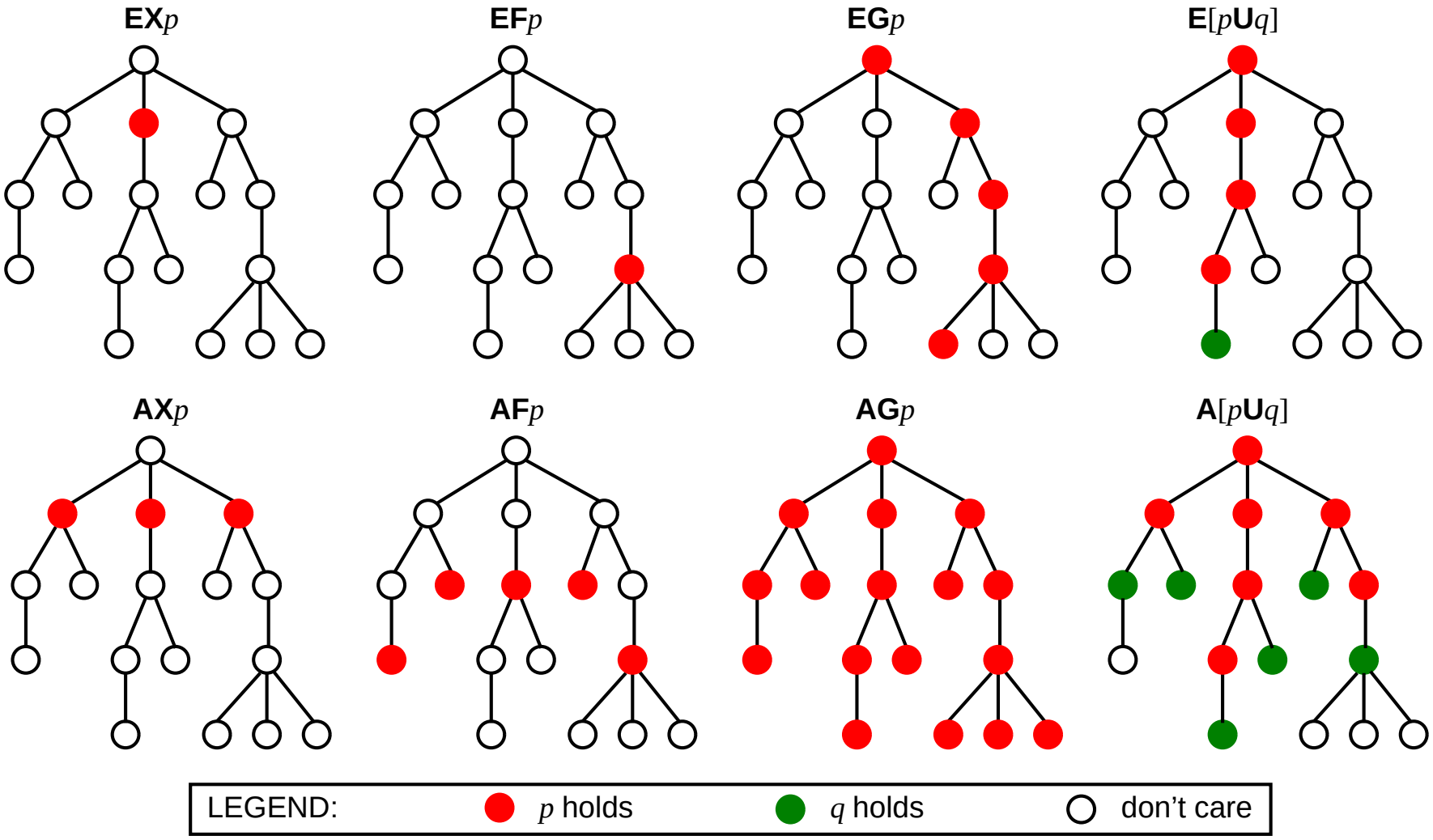
In CTL, operators occur in pairs:

- a path quantifier, E or A, must always immediately precede a temporal operator, X, F, G, U, R

Of course, CTL expressions can be nested: $p \vee \text{E}\neg p\text{U}\big(\neg p \wedge \text{AX}p\big)$

A CTL formula p identifies a set of model states (those satisfying p)

CTL semantics



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

$$\mathsf{AX}p = \neg\mathsf{EX}\neg p \qquad \mathsf{EF}p = \mathsf{E}[true\ \mathsf{U}\ p] \qquad \mathsf{E}[p\mathsf{R}q] = \neg\mathsf{A}[\neg p\mathsf{U}\neg q]$$

$$\mathsf{AF}p = \neg\mathsf{EG}\neg p \qquad \mathsf{A}[p\ \mathsf{U}\ q] = \neg\mathsf{E}[\neg q\ \mathsf{U}\ \neg p\ \wedge\ \neg q] \wedge \neg\mathsf{EG}\neg q \qquad \mathsf{A}[p\mathsf{R}q] = \neg\mathsf{E}[\neg p\mathsf{U}\neg q]$$

$$\mathsf{AG}p = \neg\mathsf{EF}\neg p$$

The EX algorithm for CTL (explicit version)

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 $\mathcal{X} \leftarrow \{\mathbf{i} \in \mathcal{X}_{reach} : p \in labels(\mathbf{i})\}$; *initialize $\mathcal{X}$ with the states satisfying p*

2 while $\mathcal{X} \neq \emptyset$ do

3 pick and remove a state $\mathbf{j}$ from $\mathcal{X}$;

4 for each $\mathbf{i} \in \mathcal{T}^{-1}(\mathbf{j})$ do *state $\mathbf{i}$ can transition to state $\mathbf{j}$*

5 $labels(\mathbf{i}) \leftarrow labels(\mathbf{i}) \cup \{\text{EX}p\}$;

The EU algorithm for CTL (explicit version)

An algorithm to label all states that satisfy $\mathsf{E}[p \mathsf{U} q]$

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

$BuildEU(p, q)$ is

1 $\mathcal{X} \leftarrow \{\mathbf{i} \in \mathcal{X}_{reach} : q \in labels(\mathbf{i})\}$; *initialize $\mathcal{X}$ with the states satisfying q*
2 for each $\mathbf{i} \in \mathcal{X}$ do
3 $\quad labels(\mathbf{i}) \leftarrow labels(\mathbf{i}) \cup \{\mathsf{E}[p \mathsf{U} q]\}$;
4 while $\mathcal{X} \neq \emptyset$ do
5 $\quad$ pick and remove a state $\mathbf{j}$ from $\mathcal{X}$;
6 $\quad$ for each $\mathbf{i} \in \mathcal{T}^{-1}(\mathbf{j})$ do *state $\mathbf{i}$ can transition to state $\mathbf{j}$*
7 $\quad\quad$ if $\mathsf{E}[p \mathsf{U} q] \notin labels(\mathbf{i})$ and $p \in labels(\mathbf{i})$ then
8 $\quad\quad\quad labels(\mathbf{i}) \leftarrow labels(\mathbf{i}) \cup \{\mathsf{E}[p \mathsf{U} q]\}$;
9 $\quad\quad\quad \mathcal{X} \leftarrow \mathcal{X} \cup \{\mathbf{i}\}$;

The EG algorithm for CTL (explicit version)

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

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

The EX algorithm for CTL (symbolic version)

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 $\mathcal{X} \leftarrow RelationalProduct(\mathcal{P}, \mathcal{T}^{-1})$; *perform one backward step in the transition relation*

2 return $\mathcal{X}$;

The EU algorithm for CTL (symbolic version)

Two algorithms to build the BDD encoding the set of states that satisfy $\mathrm{E}[p \mathrm{U} q]$

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

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

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

The EG algorithm for CTL (symbolic version)

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

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

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

Locality and the Saturation algorithm

Kronecker-consistent decomposition of a structured model

A decomposition of a discrete-state model is Kronecker-consistent if:

- $\mathcal{T}$ is disjunctively partitioned according to a set of events $\mathcal{E}$ — $\boxed{\mathcal{T}(\mathbf{i}) = \bigcup_{\alpha\in\mathcal{E}} \mathcal{T}_\alpha(\mathbf{i})}$
- $\widehat{\mathcal{X}} = \times_{L\geq k\geq 1}\mathcal{X}_k$, a global state $\mathbf{i}$ consists of L local states — $\boxed{\mathbf{i} = (i_L, \ldots, i_1)}$
- and, most importantly, we can write — $\boxed{\mathcal{T}_\alpha(\mathbf{i}) = \times_{L\geq k\geq 1}\mathcal{T}_{k,\alpha}(i_k)}$

Define the (potential) incidence matrix $\mathbf{T}[\mathbf{i},\mathbf{j}] = 1 \Leftrightarrow \mathbf{j} \in \mathcal{T}(\mathbf{i})$

$$\mathbf{T} = \sum_{\alpha\in\mathcal{E}} \mathbf{T}_\alpha = \sum_{\alpha\in\mathcal{E}} \bigotimes_{L\geq k\geq 1} \mathbf{T}_{k,\alpha}$$

We encode the next state function with $L \cdot |\mathcal{E}|$ small matrices $\mathbf{T}_{k,\alpha} \in \mathbb{B}^{|\mathcal{X}_k\times\mathcal{X}_k|}$

for Petri nets, any partition of the places into L subsets will do!

(even with inhibitor, reset, or probabilistic arcs)

Using structural information to encode $\mathcal{T}$ $(L = 5)$

$\mathcal{X}_5 = ?$ $\mathcal{X}_4 = ?$ $\mathcal{X}_3 = ?$ $\mathcal{X}_2 = ?$ $\mathcal{X}_1 = ?$

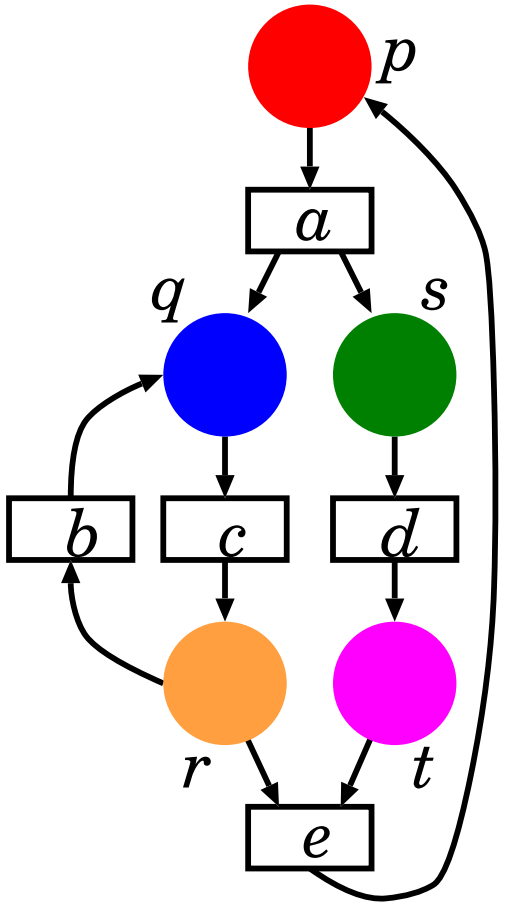
EVENTS $\longrightarrow$

LEVELS $\downarrow$

$\mathbf{T}_{5,a}: ?$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{T}_{5,e}: ?$
$\mathbf{T}_{4,a}: ?$	$\mathbf{T}_{4,b}: ?$	$\mathbf{T}_{4,c}: ?$	$\mathbf{I}$	$\mathbf{I}$
$\mathbf{I}$	$\mathbf{T}_{3,b}: ?$	$\mathbf{T}_{3,c}: ?$	$\mathbf{I}$	$\mathbf{T}_{3,e}: ?$
$\mathbf{T}_{2,a}: ?$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{T}_{2,d}: ?$	$\mathbf{I}$
$\mathbf{I}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{T}_{1,d}: ?$	$\mathbf{T}_{1,e}: ?$

$Top(a):5$ $Top(b):4$ $Top(c):4$ $Top(d):2$ $Top(e):5$

$Bot(a):2$ $Bot(b):3$ $Bot(c):3$ $Bot(d):1$ $Bot(e):1$



we determine a priori from the model whether $\mathbf{T}_{k,\alpha} = \mathbf{I}$

Kronecker encoding of $\mathcal{T}$: $\mathbf{T} = \sum_{\alpha\in\{a,b,c,d,e\}} \bigotimes_{5\geq k\geq 1} \mathbf{T}_{k,\alpha}$

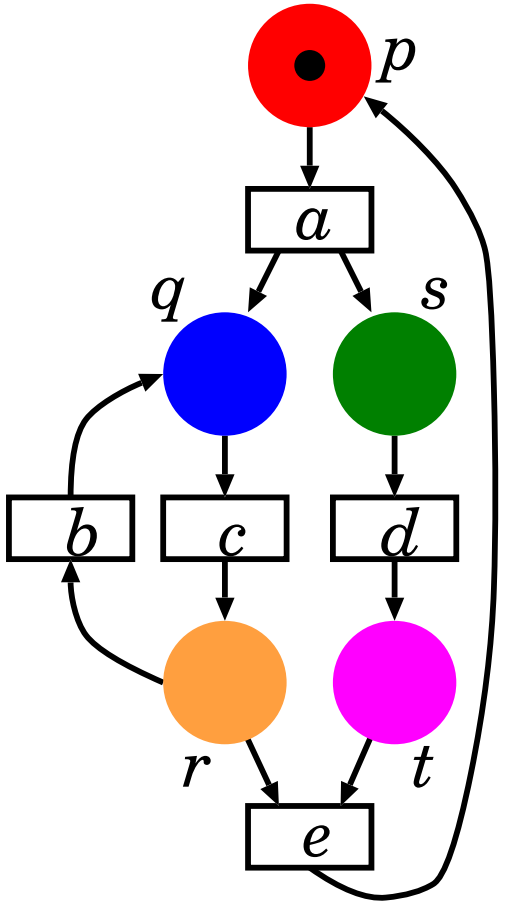
$\mathcal{X}_5 : \{p^1,p^0\}\equiv\{0,1\}$ $\mathcal{X}_4 : \{q^0,q^1\}\equiv\{0,1\}$ $\mathcal{X}_3 : \{r^0,r^1\}\equiv\{0,1\}$ $\mathcal{X}_2 : \{s^0,s^1\}\equiv\{0,1\}$ $\mathcal{X}_1 : \{t^0,t^1\}\equiv\{0,1\}$

EVENTS $\longrightarrow$

LEVELS $\downarrow$

$\mathbf{T}_{5,a}:\begin{bmatrix}0&1\\0&0\end{bmatrix}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{T}_{5,e}:\begin{bmatrix}0&0\\1&0\end{bmatrix}$
$\mathbf{T}_{4,a}:\begin{bmatrix}0&1\\0&0\end{bmatrix}$	$\mathbf{T}_{4,b}:\begin{bmatrix}0&1\\0&0\end{bmatrix}$	$\mathbf{T}_{4,c}:\begin{bmatrix}0&0\\1&0\end{bmatrix}$	$\mathbf{I}$	$\mathbf{I}$
$\mathbf{I}$	$\mathbf{T}_{3,b}:\begin{bmatrix}0&0\\1&0\end{bmatrix}$	$\mathbf{T}_{3,c}:\begin{bmatrix}0&1\\0&0\end{bmatrix}$	$\mathbf{I}$	$\mathbf{T}_{3,e}:\begin{bmatrix}0&0\\1&0\end{bmatrix}$
$\mathbf{T}_{2,a}:\begin{bmatrix}0&1\\0&0\end{bmatrix}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{T}_{2,d}:\begin{bmatrix}0&0\\1&0\end{bmatrix}$	$\mathbf{I}$
$\mathbf{I}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{T}_{1,d}:\begin{bmatrix}0&1\\0&0\end{bmatrix}$	$\mathbf{T}_{1,e}:\begin{bmatrix}0&0\\1&0\end{bmatrix}$

$Top(a):5$	$Top(b):4$	$Top(c):4$	$Top(d):2$	$Top(e):5$
$Bot(a):2$	$Bot(b):3$	$Bot(c):3$	$Bot(d):1$	$Bot(e):1$



Using structural information to encode $\mathcal{T}$ $(K = 4)$

$\mathcal{X}_4 = ?$ $\mathcal{X}_3 = ?$ $\mathcal{X}_2 = ?$ $\mathcal{X}_1 = ?$

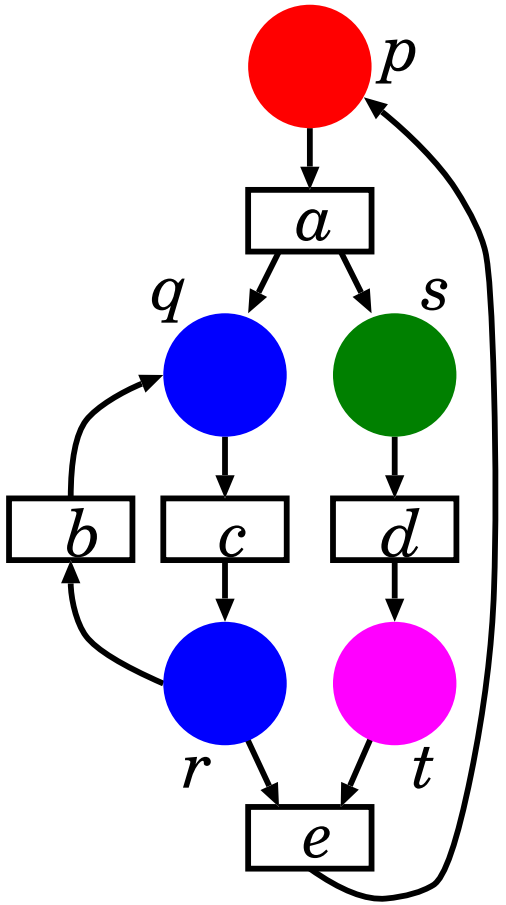
EVENTS $\longrightarrow$

LEVELS $\downarrow$

$\mathbf{T}_{4,a}$:?	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{T}_{4,e}$:?
$\mathbf{T}_{3,a}$:?	$\mathbf{T}_{3,b}$:?	$\mathbf{T}_{3,c}$:?	$\mathbf{I}$	$\mathbf{T}_{3,e}$:?
$\mathbf{T}_{2,a}$:?	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{T}_{2,d}$:?	$\mathbf{I}$
$\mathbf{I}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{T}_{1,d}$:?	$\mathbf{T}_{1,e}$:?

$Top(a):4$ $Top(b):3$ $Top(c):3$ $Top(d):2$ $Top(e):4$

$Bot(a):2$ $Bot(b):3$ $Bot(c):3$ $Bot(d):1$ $Bot(e):1$



$Top(b) = Bot(b) = Top(c) = Bot(c) = 3$: we can merge b and c into a single local event l

we determine automatically from the model whether $\mathbf{T}_{k,\alpha} = \mathbf{I}$

Kronecker encoding of $\mathcal{T}$: $\mathbf{T} = \sum_{\alpha\in\{a,b,c,d,e\}} \bigotimes_{4\geq k\geq 1} \mathbf{T}_{k,\alpha}$

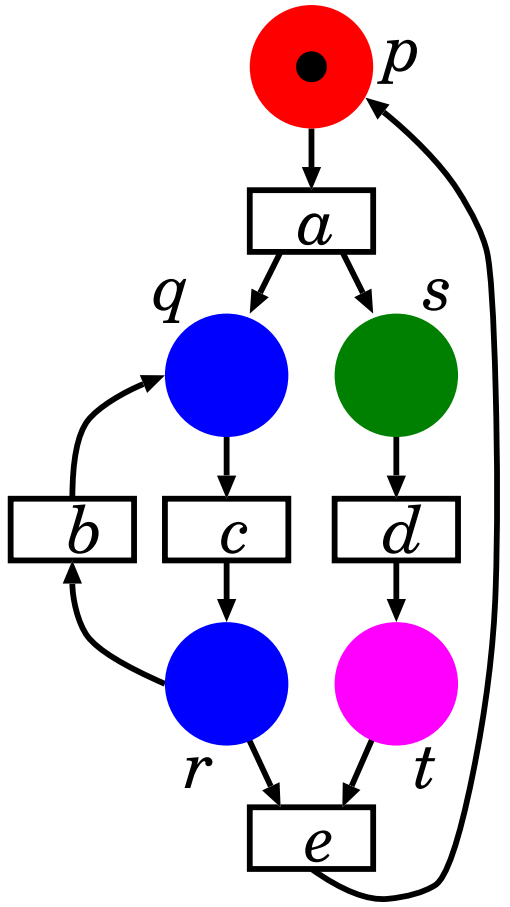
$\mathcal{X}_4 : \{p^1, p^0\} \equiv \{0,1\}$ $\mathcal{X}_3 : \{q^0r^0, q^1r^0, q^0r^1\} \equiv \{0,1,2\}$ $\mathcal{X}_2 : \{s^0, s^1\} \equiv \{0,1\}$ $\mathcal{X}_1 : \{t^0, t^1\} \equiv \{0,1\}$

EVENTS $\longrightarrow$

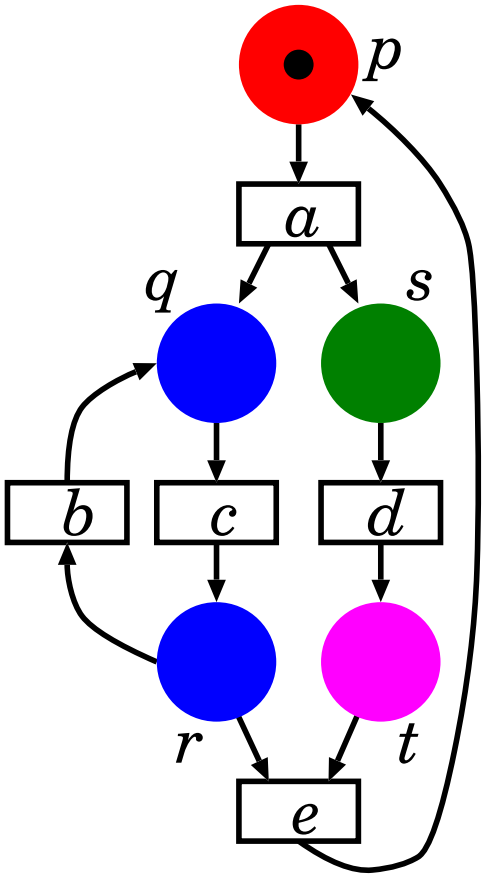
LEVELS $\downarrow$

$\mathbf{T}_{4,a}: \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{T}_{4,e}: \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$
$\mathbf{T}_{3,a}: \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\mathbf{T}_{3,b}: \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$	$\mathbf{T}_{3,c}: \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$	$\mathbf{I}$	$\mathbf{T}_{3,e}: \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$
$\mathbf{T}_{2,a}: \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{T}_{2,d}: \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$	$\mathbf{I}$
$\mathbf{I}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{T}_{1,d}: \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$	$\mathbf{T}_{1,e}: \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$

$Top(a):4$	$Top(b):3$	$Top(c):3$	$Top(d):2$	$Top(e):4$
$Bot(a):2$	$Bot(b):3$	$Bot(c):3$	$Bot(d):1$	$Bot(e):1$



The matrix $\mathbf{T}$ encoded by the Kronecker descriptor $(L = 4)$

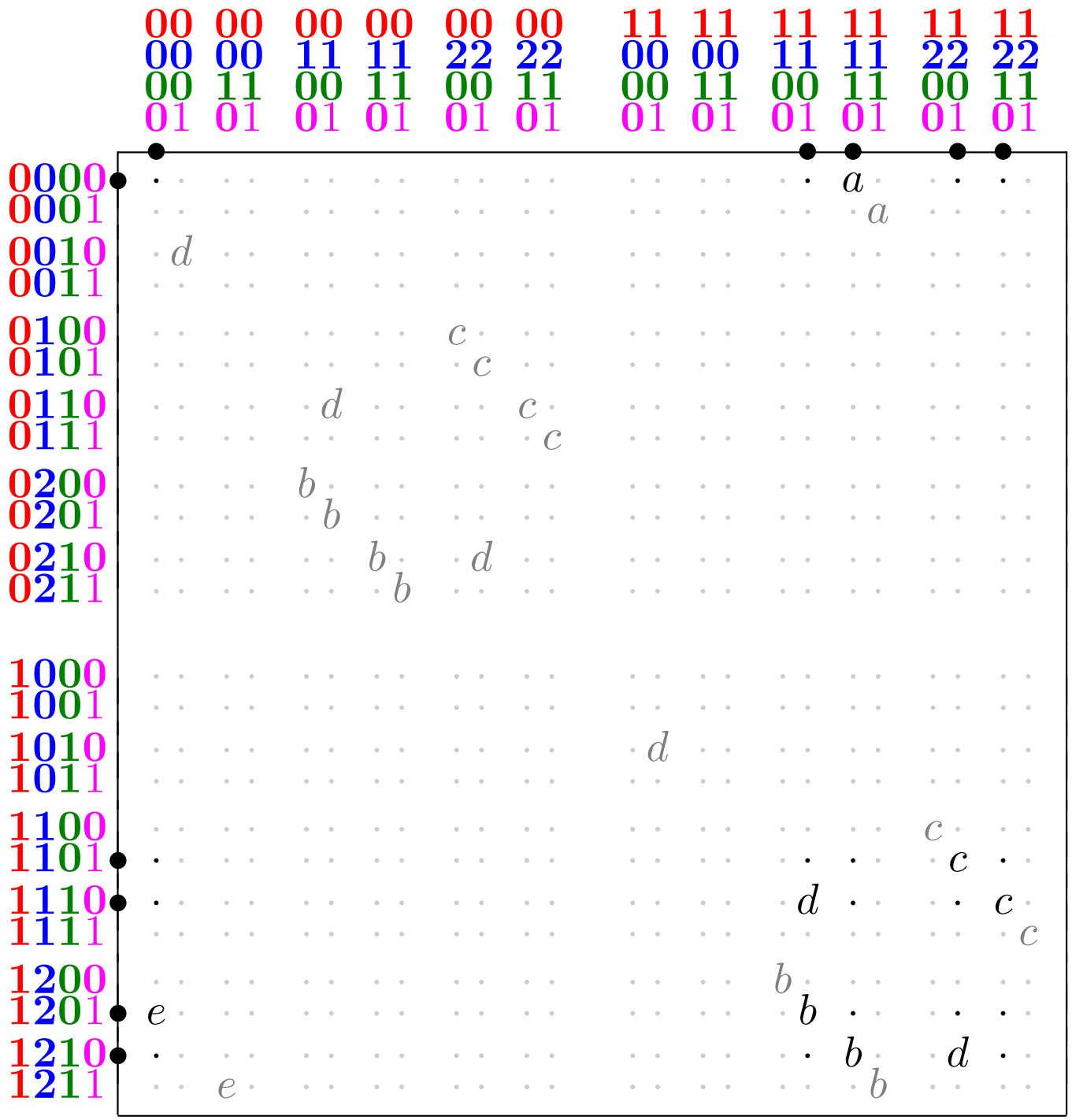


$\{p^1,p^0\} \equiv \{0,1\}$

$\{q^0r^0,q^1r^0,q^0r^1\} \equiv \{0,1,2\}$

$\{s^0,s^1\} \equiv \{0,1\}$

$\{t^0,t^1\} \equiv \{0,1\}$



Locality

The Kronecker encoding of $\mathcal{T}$ evidences locality:

- If $\mathbf{T}_{k,\alpha} = \mathbf{I}$, we say that event α and submodel k are independent
- If $\forall j_k \in \mathcal{X}_k, \mathbf{T}_{k,\alpha}[i_k, j_k] = 0$, the state of submodel k affects the enabling of event α
- If $\exists j_k \neq i_k, \mathbf{T}_{k,\alpha}[i_k, j_k] = 1$, the firing of event α can change the state of submodel k
- In the last two cases, we say that event α depends on submodel k and vice versa

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

- Let $Top(\alpha)$ and $Bot(\alpha)$ be the highest and lowest levels on which α depends
- $\{Top(\alpha), ..., Bot(\alpha)\}$ is the range (of levels) for event α, often much smaller than $\{L, ..., 1\}$

standard $2L$-level MDD encoding of $\mathcal{T}$ does not exploit locality

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

Exploiting locality

Locality takes into account the range of levels to which $\mathcal{T}_\alpha$ must be applied:

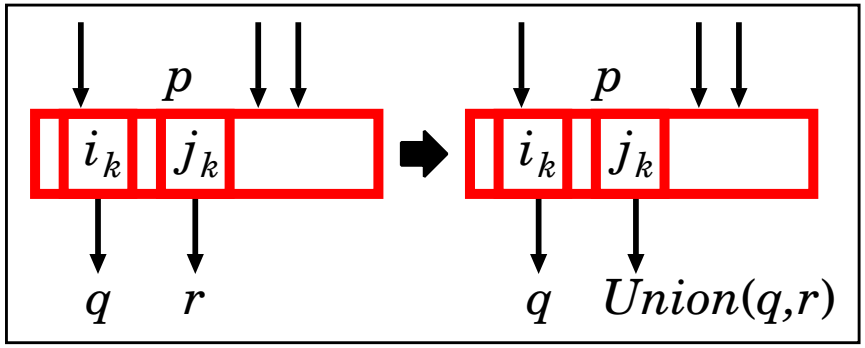
If $\mathbf{i} \in \mathcal{X}_{reach}$, $\mathbf{j} \in \mathcal{T}_\alpha(\mathbf{i})$, $Top(\alpha) = k \wedge Bot(\alpha) = h$: $\quad \mathbf{j} = (i_L, ..., i_{k+1}, j_k, ..., j_h, i_{h-1}, ..., i_1)$

In addition, it enables in-place updates of a node p at level k:

If $\mathbf{i}' = (i'_L, ..., i'_{k+1}, i_k, ..., i_1) \in \mathcal{X}_{reach}$: $\mathbf{j}' \in \mathcal{T}_\alpha(\mathbf{i}') \wedge \mathbf{j}' = (i'_L, ..., i'_{k+1}, j_k, ..., j_h, i_{h-1}, ..., i_1)$

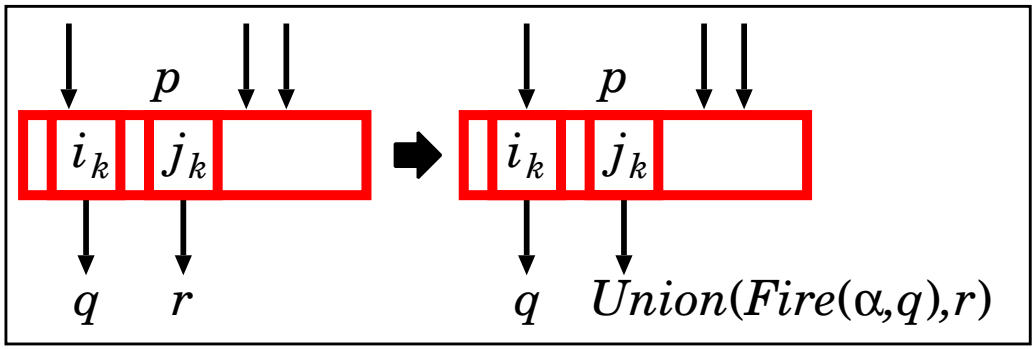
$Top(\alpha) = Bot(\alpha)$

Local event α: $i_k \xrightarrow{\alpha} j_k$



$Top(\alpha) > Bot(\alpha)$

Synchronizing event α: $(i_k, ..., i_h) \xrightarrow{\alpha} (j_k, ..., j_h)$



locality and **in-place-updates** save huge amounts of computation

Saturation: an iteration strategy based on the model structure

An MDD node p at level k is ***saturated*** if it encodes a fixed point w.r.t. any α s.t. $Top(\alpha) \leq k$

(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 s.t. $Top(\alpha) = 1$
- saturate each node at level 2: fire in them all events α s.t. $Top(\alpha) = 2$
  (if this creates nodes at level 1, saturate them immediately upon creation)
- saturate each node at level 3: fire in them all events α s.t. $Top(\alpha) = 3$
  (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 s.t. $Top(\alpha) = L$
  (if this creates nodes at levels $L-1$, $L-2$, ..., 1, saturate them immediately upon creation)

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

This can lead to enormous time and memory savings for asynchronous systems

Saturation behavior and properties

Traditional approaches apply the global next-state function $\mathcal{T}$ once to each node at each iteration and make extensive use of the unique table and operation caches

- We exhaustively fire each event α in each node p at level $k = Top(\alpha)$, from $k = 1$ up to L
- We must consider redundant nodes as well, thus we prefer quasi-reduced MDDs
- Once node p at level k is saturated, we never fire any event α with $k = Top(\alpha)$ on p again
- The recursive $Fire$ calls stop at level $Bot(\alpha)$, although the $Union$ calls can go deeper
- Only saturated nodes are placed in the unique table and in the union and firing caches
- Many (most?) nodes we insert in the MDD will still be present in the final MDD
- Firing α in p benefits from having saturated the nodes below p

usually enormous memory and time savings

but Saturation is **not** optimal for all models

Saturation pseudocode: $Saturate(L, \mathcal{X}_{init})$

mdd $Saturate$(level k, mdd p) is

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

1 if $p = \mathbf{0}$ then return $\mathbf{0}$;
2 if $p = \mathbf{1}$ then return $\mathbf{1}$;
3 if $Cache$ contains entry $\langle SaturateCODE, p : r \rangle$ then return r;
4 for $i =$ to $n_k - 1$ do $r_i \leftarrow Saturate(k-1, p[i])$; *first, be sure that the children are saturated*
5 repeat
6 choose $e \in \mathcal{E}_k$, $i, j \in \mathcal{X}_k$, s.t. $r_i \neq \mathbf{0}$ and $\mathcal{T}_e[i][j] \neq \mathbf{0}$; $\mathcal{E}_k = \{\alpha : Top(\alpha) = k\}$
7 $r_j \leftarrow Or(r_j, RelProdSat(k-1, r_i, \mathcal{T}_e[i][j]))$;
8 until $r_0, ..., r_{n_k-1}$ do not change;
9 $r \leftarrow UniqueTableInsert(k, r_0, ..., r_{n_k-1})$;
10 enter $\langle SaturateCODE, p : r \rangle$ in $Cache$;
11 return r;

mdd $RelProdSat$(level k, mdd q, mdd2 F) is

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

1 if $q = \mathbf{0}$ or $F = \mathbf{0}$ then return $\mathbf{0}$;
2 if $Cache$ contains entry $\langle RelProDsatCODE, q, F : r \rangle$ then return r;
3 for each $i, j \in \mathcal{X}_k$ s.t. $q[i] \neq \mathbf{0}$ and $F[i][j] \neq \mathbf{0}$ do $r_j \leftarrow Or(r_j, RelProdSat(k-1, q[i], F[i][j]))$;
4 $r \leftarrow Saturate(k, UniqueTableInsert(k, r_0, ..., r_{n_k-1}))$;
5 enter $\langle RelProdSatCODE, q, F : r \rangle$ in $Cache$;
6 return r.

Results: SMART vs. NuSMV

Time and memory to generate $\mathcal{X}_{reach}$ using Saturation in SMART vs. breadth–first iterations in NuSMV

N	$\lvert\mathcal{X}_{reach}\rvert$	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	—

EV$^+$MDDs and the distance function

The distance function

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

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

Build $\mathcal{X}^{[d]} = \{\mathbf{i} : \delta(\mathbf{i}) = d\}$,
for $d = 0, 1, ..., d_{max}$

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

```
1 d ← 0;   X_reach ← X_init;
2 X^[0] ← X_init;
3 repeat
4     X^[d+1] ← T(X^[d]) \ X_reach;
5     d ← d+1;   X_reach ← X_reach ∪ X^[d];
6 until X^[d] = ∅;
```

Build $\mathcal{Y}^{[d]} = \{\mathbf{i} : \delta(\mathbf{i}) \leq d\}$,
for $d = 0, 1, ..., d_{max}$

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

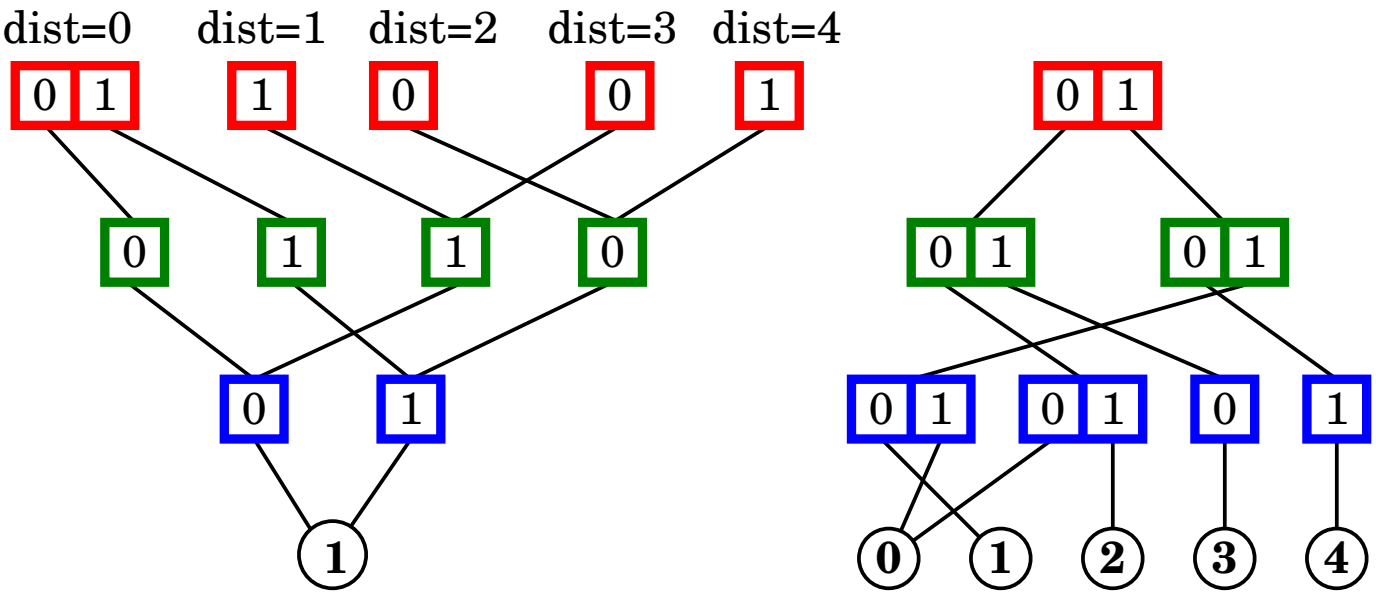
```
1 d ← 0;
2 Y^[0] ← X_init;
3 repeat
4     Y^[d+1] ← T(Y^[d]) ∪ Y^[d];
5     d ← d+1;
6 until Y^[d] = Y^[d-1];
```

This is breadth-first symbolic state space generation

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

Encoding the distance function: MDD vs. ADD (a.k.a. MTMDD)

i_3	0 0 0 0 1 1 1 1
i_2	0 0 1 1 0 0 1 1
i_1	0 1 0 1 0 1 0 1
f	0 2 3 ∞ ∞ 4 1 0



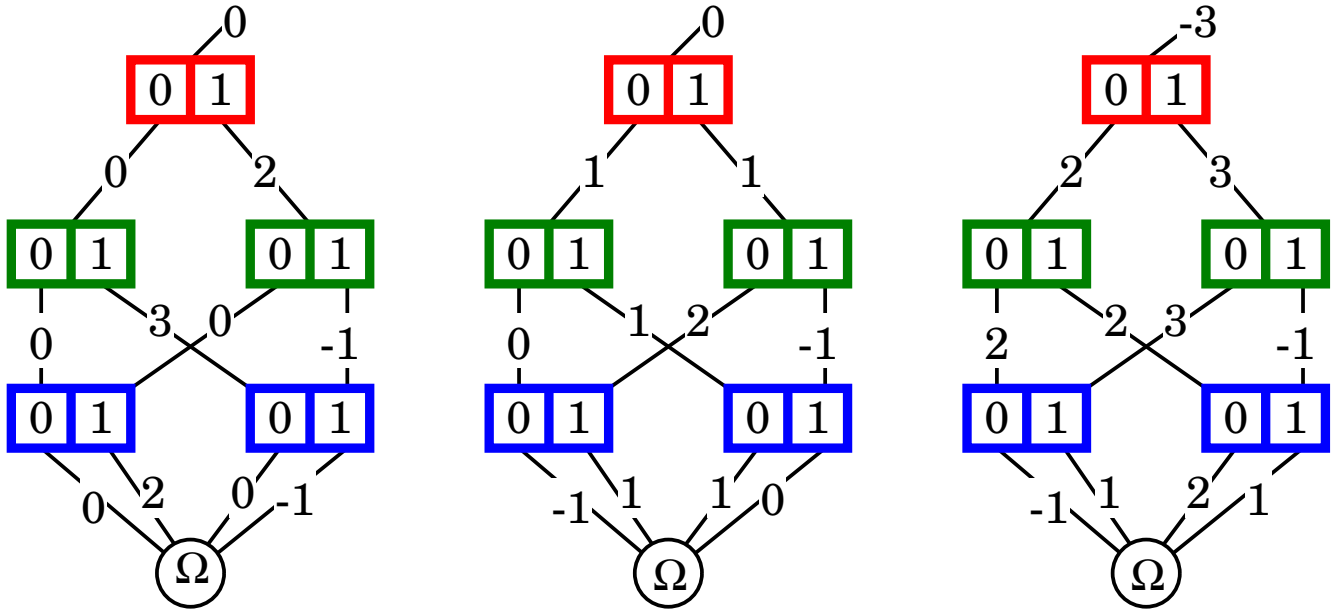
With an MDD forest: node merging can be poor at the top

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

Both approaches are explicit in the number of distinct distance values

Encoding the distance function: EVMDD [Lai et al.]

i_3	0 0 0 0 1 1 1 1
i_2	0 0 1 1 0 0 1 1
i_1	0 1 0 1 0 1 0 1
f	0 2 3 2 2 4 1 0



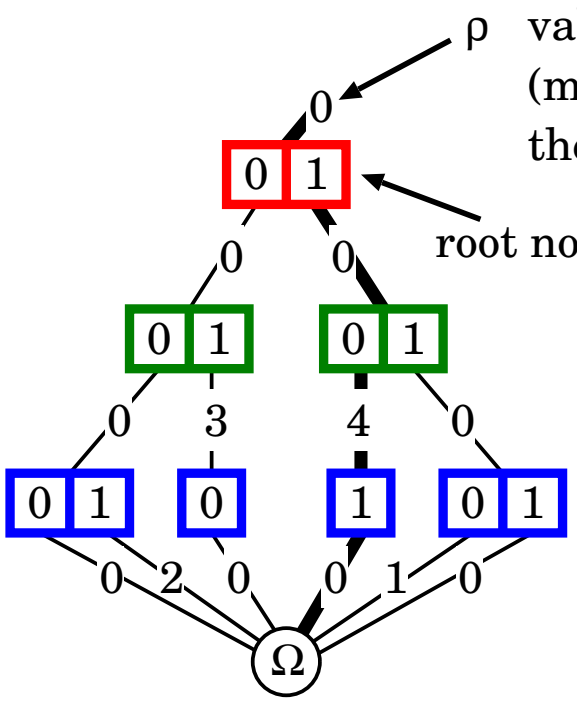
The first EVMDD is canonical (all nodes have a value 0 associated to the 0-arc)

The second and third EVMDDs are not normalized

This encoding is truly “implicit” or “symbolic”

Encoding the distance function: EV$^+$MDDs

i_3	0 0 0 0 1 1 1 1
i_2	0 0 1 1 0 0 1 1
i_1	0 1 0 1 0 1 0 1
f	0 2 3 ∞ ∞ 4 1 0



ρ value of the root edge (minimum value of the function)

root node r, at level $L = 3$

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

If (ρ, r) encodes f, then $\rho = \min\{f(\mathbf{i}) : \mathbf{i} \in \widehat{\mathcal{X}}\}$

EV$^+$MDDs can canonically represent all functions $\widehat{\mathcal{X}} \to \mathbb{Z} \cup \{\infty\}$

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

Using Saturation to compute the distance function

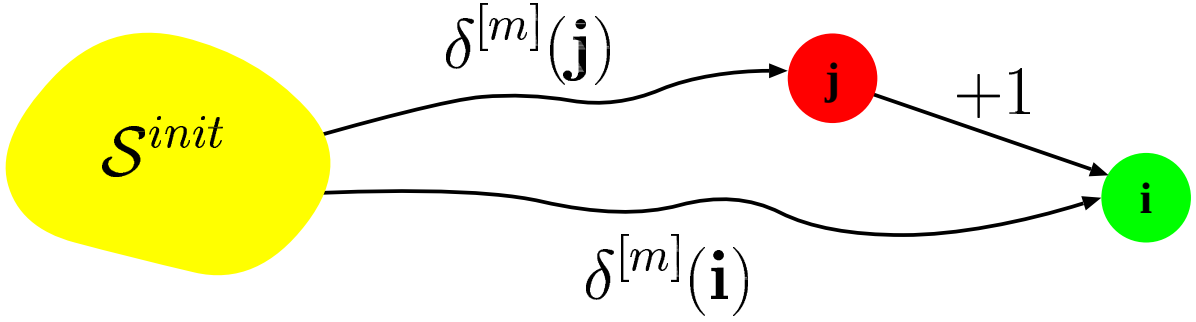
It is easy to build the distance function $\delta : \widehat{\mathcal{X}} \rightarrow \mathbb{N} \cup \{\infty\}$ using a breadth-first iteration

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

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

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

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



Results: time and memory to generate and store δ

N	$\lvert\mathcal{S}\rvert$	Time (in seconds) E_s	E_b	M_b	A_s	A_b	Final nodes E_sE_b	M_b	A_sA_b	Peak nodes E_s	E_b	M_b	A_s	A_b
Dining philosophers: $d_{max}=2N$, $L=N/2$, $\lvert\mathcal{X}_k\rvert=34$ for all k														
10	$1.9\cdot10^{6}$	0.01	0.06	0.05	0.12	0.46	**21**	255	170	**21**	605	644	238	4022
30	$6.4\cdot10^{18}$	0.02	0.86	0.70	7.39	56.80	**71**	2545	1710	**71**	7225	7364	2788	140262
1000	$9.2\cdot10^{626}$	0.48	—	—	—	—	**2496**	—	—	**2496**	—	—	—	—
Kanban system: $d_{max}=14N$, $L=4$, $\lvert\mathcal{X}_k\rvert=(N+3)(N+2)(N+1)/6$ for all k														
5	$2.5\cdot10^{6}$	0.02	0.14	0.12	0.24	1.55	9	444	133	57	1132	1156	776	13241
12	$5.5\cdot10^{9}$	0.34	4.34	3.45	11.08	129.46	16	2368	518	218	5633	5805	5585	165938
50	$1.0\cdot10^{16}$	179.48	—	—	—	—	58	—	—	2802	—	—	—	—
Flex. manuf. syst.: $d_{max}=14N$, $L=19$, $\lvert\mathcal{X}_k\rvert=N+1$ for all k except $\lvert\mathcal{X}_{17}\rvert=4$, $\lvert\mathcal{X}_{12}\rvert=3$, $\lvert\mathcal{X}_2\rvert=2$														
5	$2.9\cdot10^{6}$	0.01	0.42	0.34	0.88	11.78	149	5640	2989	211	15205	15693	4903	179577
10	$2.5\cdot10^{9}$	0.04	2.96	2.40	5.79	608.92	354	28225	11894	536	76676	78649	17885	1681625
140	$2.0\cdot10^{23}$	20.03	—	—	—	—	32012	—	—	52864	—	—	—	—
Round–robin mutex protocol: $d_{max}=8N-6$, $L=N+1$, $\lvert\mathcal{X}_k\rvert=10$ for all k except $\lvert\mathcal{X}_1\rvert=N+1$														
10	$2.3\cdot10^{4}$	0.01	0.06	0.05	0.22	0.50	92	1038	1123	107	1898	1948	1210	9245
30	$7.2\cdot10^{10}$	**0.05**	**0.95**	**0.89**	**16.04**	**224.83**	**582**	**12798**	**19495**	**637**	**24122**	**24566**	**20072**	**376609**
200	$7.2\cdot10^{62}$	1.63	—	—	—	—	20897	—	—	21292	—	—	—	—

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

A_s: ADD & Saturation A_b: ADD & breadth-first

Generating an EF trace using EV$^+$MDDs

INPUT: the MDD x encoding a set of states $\mathcal{X}$, the EV$^+$MDD (ρ, r) encoding δ

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

1. Build the EV$^+$MDD $(0, x)$ encoding $\delta_x(\mathbf{i}) = 0$ if $\mathbf{i} \in \mathcal{X}$ and $\delta_x(\mathbf{i}) = \infty$ if $\mathbf{i} \in \widehat{\mathcal{X}} \setminus \mathcal{X}$

2. Compute the EV$^+$MDD (μ, m) encoding $Max(\ (\rho, r),\ (0, x)\ )$
   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 (μ, m) a state $\mathbf{j}^{[\mu]} = (j_L^{[\mu]}, \ldots, j_1^{[\mu]})$ 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 $\mathbf{i} \in \widehat{\mathcal{X}}$ such that $\mathbf{j}^{[\nu]} \in \mathcal{T}(\mathbf{i})$ (use the backward function $\mathcal{T}^{-1}$)

   - compute $\delta(\mathbf{i})$ using (ρ, r) 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 $\mathbf{i}^*$

Markovian system models

Markov chains

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

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

$$= \Pr\left\{X(t^{[n+1]}) = \mathbf{i}^{[n+1]} \mid X(t^{[n]}) = \mathbf{i}^{[n]}\right\}$$

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

Markov property:

"given the present state, the future is independent of the past"

"the most recent knowledge about the state is all we need"

Markov chain description and analysis

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: $\mathbf{h}[\mathbf{i}] = 1 / \sum_{\mathbf{j} \in \mathcal{X}_{reach}} \mathbf{R}[\mathbf{i}, \mathbf{j}]$
- $\boldsymbol{\pi}(0)[\mathbf{i}] = \text{Pr}\{\text{chain is in state } \mathbf{i} \text{ at time } 0, \text{ i.e., initially}\}$

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

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

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

- $\boldsymbol{\pi}$ 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

Stationary solution of Markov models

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

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 j = 0 to |X_reach| − 1
3       π^(new)[j] ← h[j] · Σ_{0≤i<|X_reach|, i≠j} π^(old)[i] · R[i, j];
4     "renormalize π^(new)";
5     π^(old) ← π^(new);
6   until "converged";
7   return π^(new);
```

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

```
1   repeat
2     for j = 0 to |X_reach| − 1
3       π[j] ← h[j] · Σ_{0≤i<|X_reach|, i≠j} π[i] · R[i, j];
4     "renormalize π";
5   until "converged";
6   return π;
```

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

State indexing options: potential $\widehat{\psi}$ vs. actual ψ

For Markov analysis, we can generate $\mathcal{X}_{reach}$ first, using $\mathcal{X}_{init}$ and $\mathcal{T} : \widehat{\mathcal{X}} \to 2^{\widehat{\mathcal{X}}}$

Once we know $\mathcal{X}_{reach}$:

- We can restrict $\mathcal{T}$ to $\mathcal{T} : \mathcal{X}_{reach} \to 2^{\mathcal{X}_{reach}}$ (if needed for further logical analysis)
- We can store $\widehat{\mathbf{R}} : \widehat{\mathcal{X}} \times \widehat{\mathcal{X}} \to \mathbb{R}$ or $\mathbf{R} : \mathcal{X}_{reach} \times \mathcal{X}_{reach} \to \mathbb{R}$
- We can choose algorithms that use $\widehat{\boldsymbol{\pi}} : \widehat{\mathcal{X}} \to \mathbb{R}$ or $\boldsymbol{\pi} : \mathcal{X}_{reach} \to \mathbb{R}$

Strictly **explicit** methods: using actual $\mathbf{R}$ and $\boldsymbol{\pi}$ works best

Strictly **implicit** methods: decision diagrams usually don't work well to store $\widehat{\boldsymbol{\pi}}$ or $\boldsymbol{\pi}$

Implicit methods have tradeoffs:

- Storing $\boldsymbol{\pi}$ instead of $\widehat{\boldsymbol{\pi}}$ is often unavoidable if we employ a full vector and $|\widehat{\mathcal{X}}| \gg |\mathcal{X}_{reach}|$
- Symbolic storage of $\widehat{\mathbf{R}}$ is usually cheaper than that of $\mathbf{R}$ in terms of memory requirements
- However, using $\widehat{\mathbf{R}}$ in conjunction with $\boldsymbol{\pi}$ complicates indexing...
- ...forcing us to store $\psi : \widehat{\mathcal{X}} \to \{0, 1, \ldots, |\mathcal{X}_{reach}| - 1\} \cup \{\text{null}\}$, hence $\mathcal{X}_{reach}$

MxD-based vector-matrix multiplication algorithm

real$[n]$ $VectorMatrixMult$(real$[n]$ $\mathbf{x}$, mxd_node A, evmdd_node ψ) is $\quad n = |\mathcal{X}_{reach}|$

```
local natural s;                                          -- state index in x
local real[n] y;
local sparse_real c;
1 s ← 0;
2 for each j = (j_L, ..., j_1) ∈ X_reach in lexicographic order do    -- s = ψ(j)
3     c ← GetCol(L, A, ψ, j_L, ..., j_1);                 -- build column j of A using sparse storage
4     y[s] ← ElementWiseMult(x, c);                       -- x uses full storage, c uses sparse storage
5     s ← s + 1;
6 return y;
```

sparse_real $GetCol$(level k, mxd_node M, evmdd_node ϕ, natural $j_k, ..., j_1$) is

```
local sparse_real c, d;
1 if k = 0 then return [1];                               -- a vector of size one, with its entry set to 1
2 if Cache contains entry ⟨GetColCODE, M, φ, j_k, ..., j_1 : c⟩ then return c;
3 c ← 0;                                                  -- initialize the result to all zero entries
4 for each i_k ∈ X_k such that M[i_k, j_k].val ≠ 0 and φ[i_k].val ≠ ∞ do
5     d ← GetCol(k − 1, M[i_k, j_k].child, φ[i_k].child, j_{k−1}, ..., j_1);
6     for each i such that d[i] ≠ 0 do
7         c[i + φ[i_k].val] ← c[i + φ[i_k].val] + M[i_k, j_k].val · d[i];
8 enter ⟨GetColCODE, M, φ, j_k, ..., j_1 : c⟩ in Cache;
9 return c;
```

SMART

What is SMART?

- A package integrating logical and stochastic modeling formalisms into a single environment
- Multiple parametric models expressed in different formalisms can be combined in a study
- Easy addition of new formalisms and solution algorithms
- For the study of logical behavior:
  - explicit (BFS exploration) state-space generation [Tools97]
  - implicit (symbolic MDD Saturation) state-space generation [TACAS01,03]
  - next-state function with Kronecker products or matrix diagrams [PNPM99, IPDS01]
  - Saturation-based CTL symbolic model checking [CAV03]
- For the study of stochastic and timing behavior
  - explicit (sparse storage) numerical solution of CTMCs and DTMCs
  - implicit (Kronecker) numerical solution of CTMCs [INFORMSJC00]
  - structural-based approximations of large CTMC models [SIGMETRICS00]
  - explicit numerical solution of semi-regenerative models [PNPM01]
  - simulation of arbitrary models
  - regenerative simulation with automatic detection of regeration points [PMCCS03]
  - structural caching to speed up simulation [PMCCS03]

Strongly-typed, computation-on-demand, language.

Five types of basic statements . . .

- **Declaration statements** declare functions over some arguments (including constants)
- **Definition statements** declare functions and specify how to compute their value
- **Model statements** define parametric models (declarations, specifications, measures)
- **Expression statements** print values (may have side-effects)
- **Option statements** modify the behavior of SMART (precision, verbosity level)

Two compound statements that can be arbitrarily nested

- `for` statements define arrays or repeatedly evaluate parametric expressions
  Useful for parametric studies
- `converge` **statements** specify numerical fixed-point iterations
  Useful for approximate performance or reliability studies
  Cannot appear within the declaration of a model

Basic predefined types are available in SMART

- `bool`: the values `true` or `false`
- `int`: integers (machine-dependent)
- `bigint`: arbitrary-size integers
- `real`: floating-point values (machine-dependent)
- `string`: character-array values

```
bool c := 3 - 2 > 0;
int i := -12;
bigint i := 1234567890123456789*2;
real x := sqrt(2.3);
string s := "Monday";
```

Composite types can be defined

- aggregate: analogous to the Pascal “record” or C “struct”
- set: collection of homogeneous objects
- array: collection of homogeneous objects indexed by set elements

```
p:t:3
{1..8,10,25,50}
```

A type can be further modified by a nature describing stochastic characteristics

- `const`: (the default) a non-stochastic quantity
- `ph`: a random variable with discrete or continuous phase-type distribution
- `rand`: a random variable with arbitrary distribution
- `proc`: a random variable that depends on the state of a model at a given time

Predefined formalism types can be used to define discrete state models (logical or stochastic)

Function declarations

Objects in SMART are functions, possibly recursive, that can be overloaded

```
real pi := 3.14;  // a constant, i.e., a 0-argument function

bool close(real a, real b) := abs(a-b) < 0.00001;

int  pow(int  b, int e) := cond(e==1, b, b*pow(b,e-1));

real pow(real b, int e) := cond(e==1, b, b*pow(b,e-1));
```

```
pow(5,3);     // expression, not declaration, prints 125, int

pow(0.5,3); // expression, not declaration, prints 0.125, real
```

Arrays

Arrays are declared using a `for` statement

An array's dimensionality is determined by the enclosing iterators

Indices along each dimension belong to a finite set

We can define arrays with `real` indices

```
for (int i in {1..5}, real r in {1..i..0.5}) {
  real res[i][r]:= MyModel(i,r).out1;
}
```

`res` is not a "rectangular" array of values:

- `res[1][1.0]`
- `res[2][1.0]`, `res[2][1.5]`, `res[2][2.0]`
- …
- `res[5][1.0]`, `res[5][1.5]`, …, `res[5][5.0]`

State-space generation and storage: model partition

SMART uses the `#StateStorage` option to choose between

- explicit techniques that store each state individually
  (`AVL`, `SPLAY`, `HASHING`)
  - no restrictions on the model
  - require time and memory at least linear in the number of reachable states
- implicit techniques that employ MDDs to symbolically store sets of states
  (`MDD_LOCAL_PREGEN`, `MDD_SATURATION_PREGEN`, `MDD_SATURATION`)
  - normally much more efficient
  - require a Kronecker-consistent partition of the model, automatically checked by SMART
    (global model behavior is the logical product of each submodel's local behavior)

A PN partition is specified by giving class indices (contiguous, positive integers) to places:

```
partition(2:p); partition(1:r); partition(1:t, 2:q, 1:s);
```

or by simply enumerating (without index information) the places in each class

```
partition(p:q, r:s:t);
```

State-space generation and storage: dining philosopers


```
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(1+div(i,2):idle[i]:waitL[i]:waitR[i]:hasL[i]:hasR[i]:fork[i]);
    init(idle[i]:1, fork[i]:1);
    trans  Go[i],GetL[i],GetR[i],Stop[i];
  }
  for (int i in {0..N-1}) {
    arcs(idle[i]:Go[i], Go[i]:waitL[i], Go[i]:waitR[i],
         waitL[i]:GetL[i], waitR[i]:GetR[i],
         fork[i]:GetL[i], fork[mod(i+1,N)]:GetR[i],
         GetL[i]:hasL[i], GetR[i]:hasR[i], hasL[i]:Stop[i], hasR[i]:Stop[i],
         Stop[i]:idle[i], Stop[i]:fork[i], Stop[i]:fork[mod(i+1, N)]);
  }
  bigint n_s := num_states(false);
};
# StateStorage MDD_SATURATION
print("The model has ", phils(read_int("N")).n_s, " states.\n");
```

Number of philosophers	States $\lvert\mathcal{S}\rvert$	MDD Nodes		Mem. (bytes)		CPU (secs)
		Final	Peak	Final	Peak	
100	4.97×10^{62}	197	246	30,732	38,376	0.04
1,000	9.18×10^{626}	1,997	2,496	311,532	389,376	0.45
10,000	4.26×10^{6269}	19,997	24,496	3,119,532	3,821,376	314.13

CTL model checking: operations

An object of type `stateset`, a set of global model states, is stored as an MDD

All MDDs for a model instance are stored in one MDD forest with shared nodes for efficiency

- **Atom builders:**
  - `nostates`, returns the empty set
  - `initialstate`, returns the initial state or states of the model
  - `reachable`, returns the set of reachable states in the model
  - `potential(`e`)`, returns the states of $\widehat{\mathcal{X}}$ satisfying condition e
- **Set operators:**
  - `union(`$\mathcal{P}$, $\mathcal{Q}$`)`, returns $\mathcal{P} \cup \mathcal{Q}$
  - `intersection(`$\mathcal{P}$, $\mathcal{Q}$`)`, returns $\mathcal{P} \cap \mathcal{Q}$
  - `complement(`$\mathcal{P}$`)`, returns $\widehat{\mathcal{X}} \setminus \mathcal{P}$
  - `difference(`$\mathcal{P}$, $\mathcal{Q}$`)`, returns $\mathcal{P} \setminus \mathcal{Q}$
  - `includes(`$\mathcal{P}$, $\mathcal{Q}$`)`, returns `true` iff $\mathcal{P} \supseteq \mathcal{Q}$
  - `eq(`$\mathcal{P}$, $\mathcal{Q}$`)`, returns `true` iff $\mathcal{P} = \mathcal{Q}$

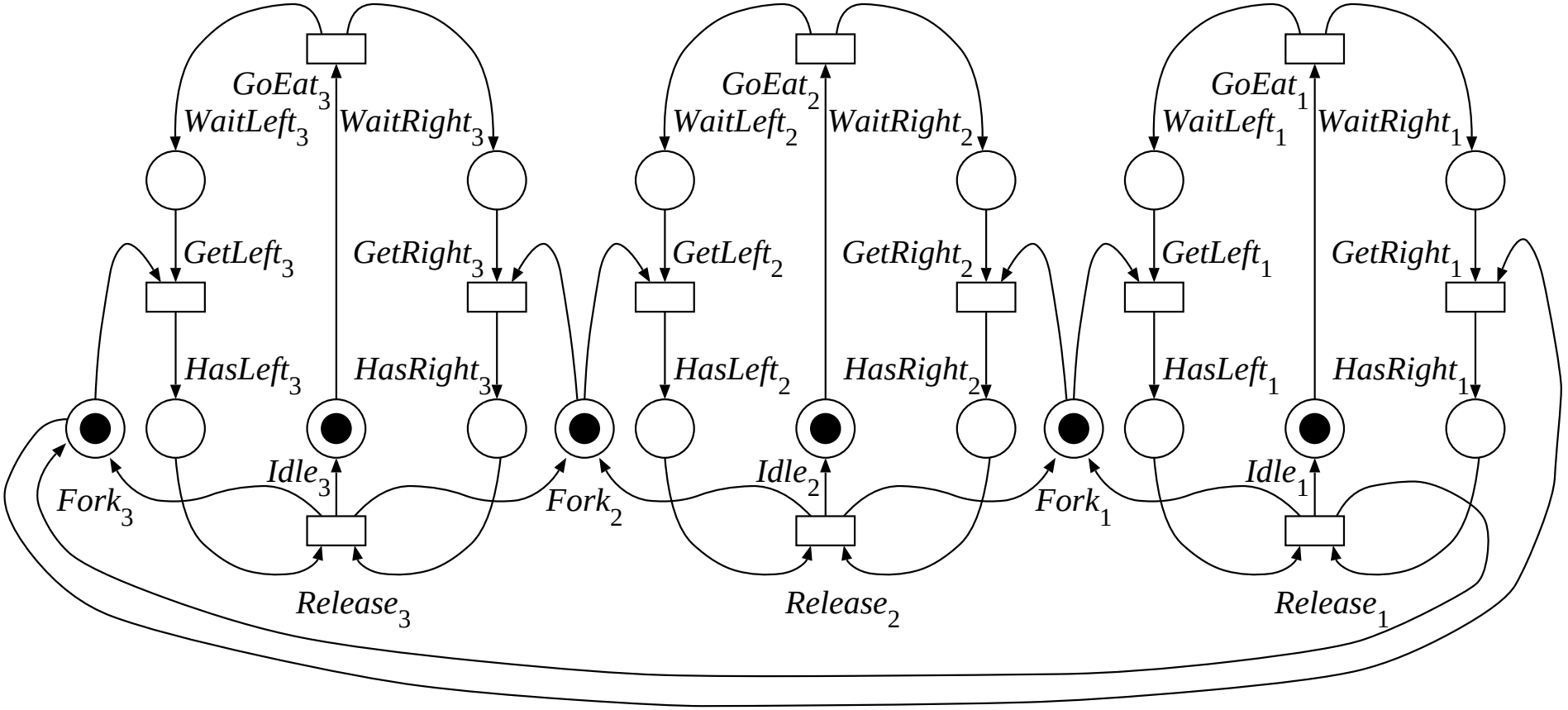
CTL model checking: operations (cont.)

- **Temporal logic operators (future and past CTL operators):**
  - `EX(`$\mathcal{P}$`)` and `EXbar(`$\mathcal{P}$`)`, `AX(`$\mathcal{P}$`)` and `AXbar(`$\mathcal{P}$`)`
  - `EF(`$\mathcal{P}$`)` and `EFbar(`$\mathcal{P}$`)`, `AF(`$\mathcal{P}$`)` and `AFbar(`$\mathcal{P}$`)`
  - `EG(`$\mathcal{P}$`)` and `EGbar(`$\mathcal{P}$`)`, `AG(`$\mathcal{P}$`)` and `AGbar(`$\mathcal{P}$`)`
  - `EU(`$\mathcal{P}$`,` $\mathcal{Q}$`)` and `EUbar(`$\mathcal{P}$`,` $\mathcal{Q}$`)`, `AU(`$\mathcal{P}$`,` $\mathcal{Q}$`)` and `AUbar(`$\mathcal{P}$`,` $\mathcal{Q}$`)`
- **Execution trace output:**
  - `EFtrace(`$\mathcal{R}$`,` $\mathcal{P}$`)`, prints a witness for `EF(`$\mathcal{P}$`)` starting from a state in $\mathcal{R}$
  - `EGtrace(`$\mathcal{R}$`,` $\mathcal{P}$`)`, prints a witness for `EG(`$\mathcal{P}$`)` starting from a state in $\mathcal{R}$
  - `EUtrace(`$\mathcal{R}$`,` $\mathcal{P}$`,` $\mathcal{Q}$`)`, prints a witness for `EU(`$\mathcal{P}$`,` $\mathcal{Q}$`)` starting from a state in $\mathcal{R}$
  - `dist(`$\mathcal{P}$`,` $\mathcal{Q}$`)`, returns the length of a shortest path from $\mathcal{P}$ to $\mathcal{Q}$
- **Utility functions:**
  - `card(`$\mathcal{P}$`)`, returns the number of states in $\mathcal{P}$ (as a `bigint`)
  - `printset(`$\mathcal{P}$`)`, prints the states in $\mathcal{P}$ (up to a given maximum)

SMART uses EV$^+$MDDs for efficient witness generation...

...and Saturation for efficient CTL model checking

Example: the dining philosophers (Petri net)



N subnets connected in a circular fashion

Example: the dining philosophers (SMART code)


```
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;
```

Example: the dining philosophers (results)


```
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 Wait
State 1 : { WaitL[0]:1 HasR[0]:1 WaitL[1]:1 HasR[1]:1 WaitL[2]:1 HasR[2]:1 Wait
true
Done.
```