Petri net analysis using decision diagrams

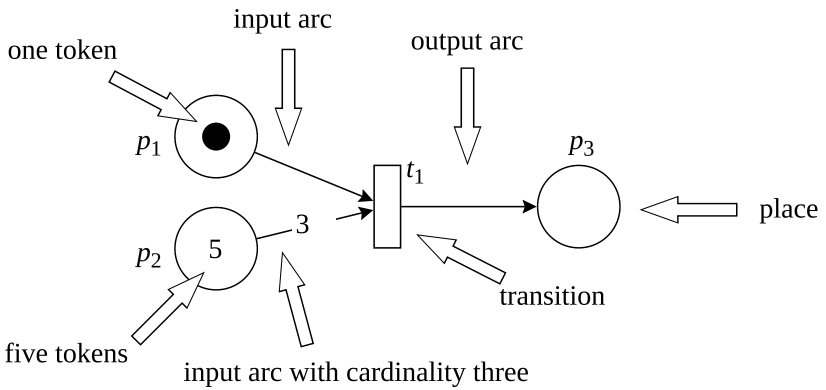
Gianfranco Ciardo

Department of Computer Science and Engineering

University of California, Riverside

State-space generation of Petri nets

Graphical representation of a Petri net



Petri nets

A Petri net is a tuple $(\mathcal{P}, \mathcal{E}, \mathbf{D}^-, \mathbf{D}^+, \mathbf{x}_{init})$ where:

- $\mathcal{P}$ — set of places, drawn as circles
- $\mathcal{E}$ — set of transitions drawn as rectangles
- $\mathbf{D}^- : \mathcal{P} \times \mathcal{E} \to \mathbb{N}$ — input arc cardinalities
- $\mathbf{D}^+ : \mathcal{P} \times \mathcal{E} \to \mathbb{N}$ — output arc cardinalities
- $\mathbf{x}_{init} \in \mathbb{N}^{|\mathcal{P}|}$ — initial state, or marking

with $\mathcal{P} \cap \mathcal{E} = \emptyset$

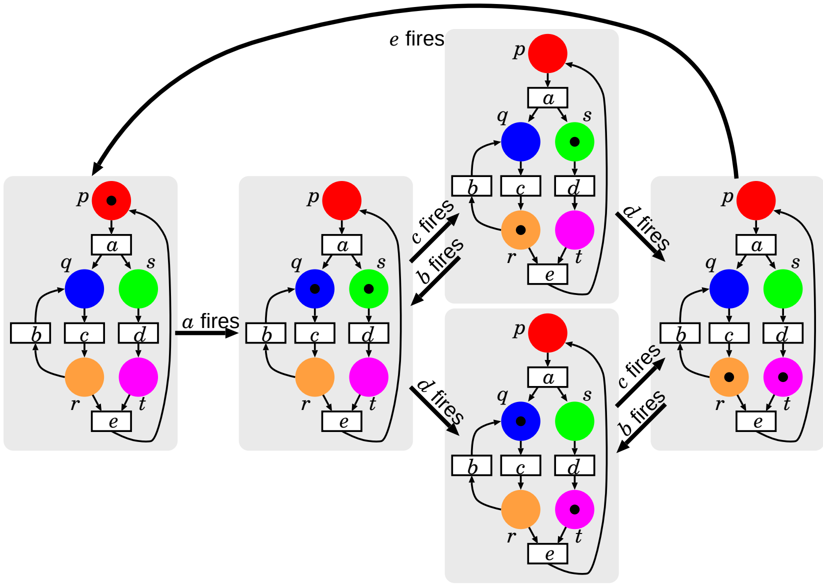
Condition for transition α to be enabled in state $\mathbf{i} \in \mathbb{N}^{|\mathcal{P}|}$: $\quad \alpha \in \mathcal{E}(\mathbf{i}) \;\Leftrightarrow\; \forall p \in \mathcal{P},\ \mathbf{D}^-_{p,\alpha} \leq \mathbf{i}_p$

A transition α enabled in state $\mathbf{i}$ can fire: $\quad \mathbf{i} \xrightarrow{\alpha} \mathbf{j} \;\Leftrightarrow\; \forall p \in \mathcal{P},\ \mathbf{j}_p = \mathbf{i}_p - \mathbf{D}^-_{p,\alpha} + \mathbf{D}^+_{p,\alpha}$

The next-state function $\mathcal{N}$ satisfies $\quad \mathbf{j} \in \mathcal{N}(\mathbf{i}) \;\Leftrightarrow\; \exists \alpha \in \mathcal{E},\ \mathbf{j} \in \mathcal{N}_\alpha(\mathbf{i}) \;\Leftrightarrow\; \exists \alpha \in \mathcal{E},\ \mathbf{i} \xrightarrow{\alpha} \mathbf{j}$

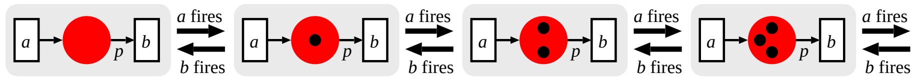
The state space, or reachability set, $\mathcal{X}_{reach}$ is defined as usual

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

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



$\mathcal{X}_{reach}$ contains an infinite number of states regardless of the initial state $\mathbf{x}_{init} = (N)$

Why is state-space generation important?

- State-space generation is an essential step in logical model analysis
  - Can be used to discover the potential states (the possible range of a variable)
  - Discovers the actual states of the system (the possible combinations of variable values)
- State-space generation is enough to answer safety queries
  - Can we reach a "bad" state?
  - Is it true that $VAL < 0$ whenever $FLAG$ is raised?
- State-space generation is similar to other temporal logic queries
  - Is it possible to reach a state where $VAL < 0$ from a state where $VAL > 0$? (EF)
  - Is it true that, if $VAL > 0$, it must become 0 before it can become negative? (EU)

symbolic methods based on decision diagrams help immensely...

...but we still are (and always will be) memory and time bound

State-by-state (explicit) generation algorithm

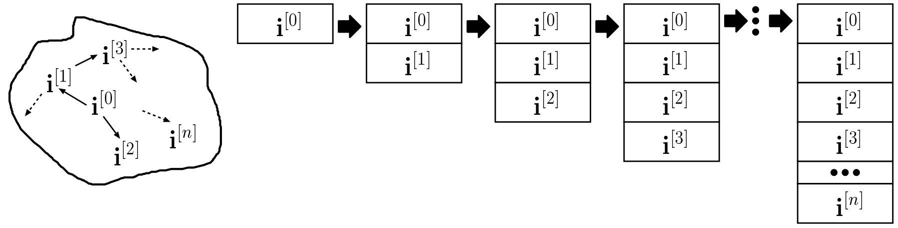
ExploreExplicit($\mathcal{X}_{init}$: set of states, $\mathcal{N}$: next-state functionn) : set of states
local $\mathcal{U}$, $\mathcal{X}_{reach}$: set of states;
local $\mathbf{i}$, $\mathbf{j}$: state;
1 $\mathcal{X}_{reach} \leftarrow \emptyset$; — *$\mathcal{X}_{reach}$ contains the known states 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 to $\mathcal{X}_{reach}$;
5 for each $\mathbf{j} \in \mathcal{N}(\mathbf{i})$ do
6 if $\mathbf{j} \notin \mathcal{X}_{reach} \cup \mathcal{U}$ 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 $\mathcal{X}_{reach}$;

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

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

State-by-state (explicit) generation of $\mathcal{X}_{reach}$

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



with an explicit data structure

memory requirements increase monotonically during generation

they are proportional to $|\mathcal{X}_{reach}|$ at the end

Explicit vs. implicit: what's the difference?

Several informal concepts of "explicit" and "implicit" (a.k.a. "symbolic") have been used

We adopt the following informal definitions

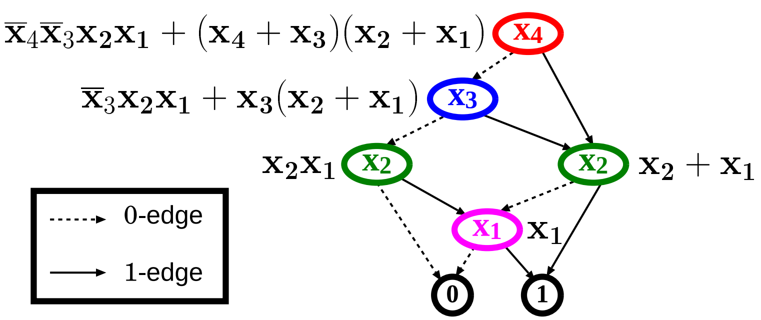
- Explicit data structure: each state requires a different memory location (bit, byte, word, array, etc.)
  $\Rightarrow$ $O(|\mathcal{X}_{reach}|)$ memory
- Explicit algorithm: states are manipulated one by one
  $\Rightarrow$ $O(|\mathcal{X}_{reach}|)$ or maybe $O(|\mathcal{X}_{reach}| \cdot \log|\mathcal{X}_{reach}|)$ time
  Memory requirements increase linearly as new states are found

- Implicit data structure: each memory location **may** store information about multiple states
  $\Rightarrow$ $O(|\mathcal{X}_{reach}|)$ memory only in the worst case
- Implicit algorithm: states are manipulated one set at a time
  $\Rightarrow$ $O(|\mathcal{X}_{reach}|)$ or maybe $O(|\mathcal{X}_{reach}| \cdot \log|\mathcal{X}_{reach}|)$ time only in the worst case

  Memory requirements grow and shrink as new states are found, **peak** not usually at the end

(Reduced ordered) binary decision diagrams (BDDs)

"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 $\quad f : \mathbb{B}^L \rightarrow \mathbb{B}$

For the root node, $f(\mathbf{x}_4\!=\!0, \mathbf{x}_3\!=\!1, \mathbf{x}_2\!=\!1, \mathbf{x}_1\!=\!0) = f(\mathbf{x}_4\!=\!0, \mathbf{x}_3\!=\!1, \mathbf{x}_2\!=\!1, \mathbf{x}_1\!=\!1) = 1$

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 $\qquad \mathbf{0}.lvl = \mathbf{1}.lvl = 0$
- A nonterminal node p is at a level k, with $L \geq k \geq 1$ $\qquad p.lvl = k$
- 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; $\qquad p[0].lvl < p.lvl,\ p[1].lvl < p.lvl$
- 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 $p[0] = p[1]$

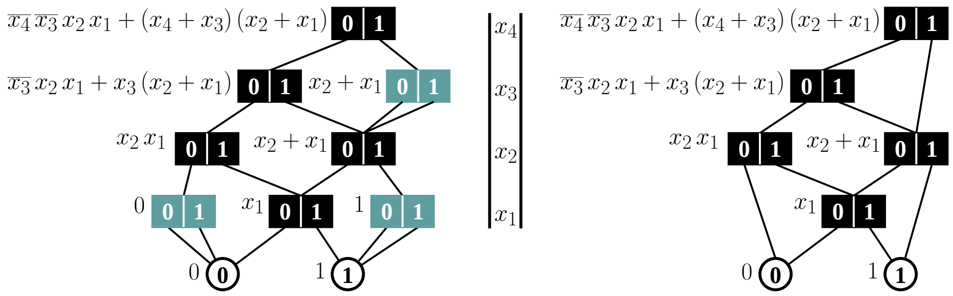
Both versions are **canonical**, thus, if functions f and g are encoded using BDDs,

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

$||f||$ = number of nodes in the BDD encoding f

$||f||_k$ = number of nodes at level k in the BDD encoding f

Quasi-reduced vs. fully-reduced BDDs



Fully-reduced BDDs: each node in the BDD encodes a different function

Quasi-reduced BDDs: each node at a given level of the BDD encodes a different function

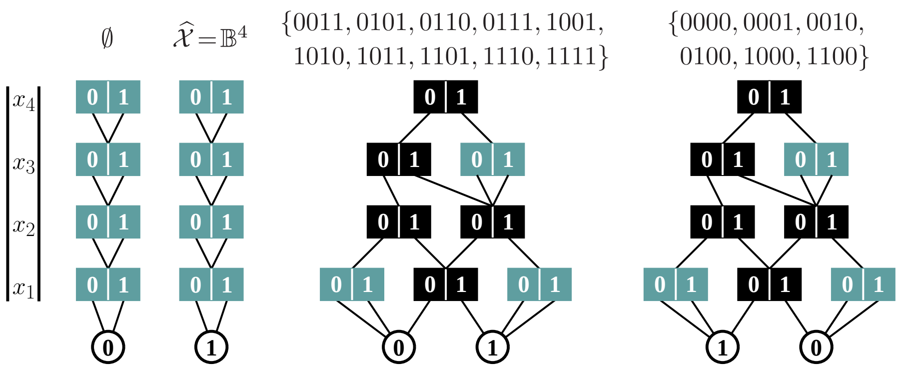
Using BDDs to encode sets

We can encode a set $\mathcal{Y} \subseteq \mathbb{B}^L$ as a BDD p through its characteristic function:

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

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

Indeed, any set requires as many nodes as its complement:



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

fully-reduced version

```
bdd Union(bdd p, bdd q) is
local bdd r;
if p = 0 or q = 1 then return q;
if q = 0 or p = 1 then return p;
if p = q then return p;
if Cache contains entry ⟨UnionCODE, {p, q} : r⟩ then return r;
if p.lvl = q.lvl then
    r ← UniqueTableInsert(p.lvl, Union(p[0], q[0]), Union(p[1], q[1]));
else if p.lvl > q.lvl then
    r ← UniqueTableInsert(p.lvl, Union(p[0], q), Union(p[1], q));
else since p.lvl < q.lvl then
    r ← UniqueTableInsert(q.lvl, Union(p, q[0]), Union(p, q[1]));
enter ⟨UnionCODE, {p, q} : r⟩ in Cache;
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 $O($product of the numbers of nodes in p and $q)$

Computing the relational product symbolically

Given an L-level BDD on $(x_L, ..., x_1)$ rooted at p_* encoding a set $\mathcal{Y} \subseteq \widehat{\mathcal{X}}$

Given a $2L$-level BDD on $(x_L, x'_L ..., x_1, x'_1)$ rooted at r_* encoding a function $\mathcal{N} : \widehat{\mathcal{X}} \to 2^{\widehat{\mathcal{X}}}$

$RelationalProduct(p_*, r_*)$ returns the root of the BDD encoding the set

$$\{\mathbf{j} : \exists \mathbf{i} \in \mathcal{Y} \ \wedge \ \mathbf{j} \in \mathcal{N}(\mathbf{i})\}$$

quasi-reduced version

```
bdd RelationalProduct(bdd p, bdd2 r) is
local  bdd  q, q1, q2;
if p = 0 or r = 0 then return 0;
if p = 1 and r = 1 then return 1;
if Cache contains entry ⟨RelationalProductCODE, p, r : q⟩ then return q;
q0 ← Union(RelationalProduct(p[0], r[0][0]), RelationalProduct(p[1], r[1][0]));
q1 ← Union(RelationalProduct(p[0], r[0][1]), RelationalProduct(p[1], r[1][1]));
q ← UniqueTableInsert(p.lvl, q0, q1);
enter ⟨RelationalProductCODE, p, r : q⟩ in Cache;
return q;
```

Properties of fully-reduced ordered BDDs

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

- The variable ordering affects the size of the BDD, consider $x_L = y_L \wedge \cdots \wedge x_1 = y_1$
  - 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 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 BDD size is an NP-complete problem

Efficiency of BDDs

The efficient manipulation of BDDs relies on the idea of dynamic programming

More specifically, on the use of an operation cache

Given enough memory for cache entries, we never recompute an operation on the same operands

The operation cache is implemented as a hash table with entries of the form ⟨ key : result ⟩

- given the key $(operator, operand, ..., operand)$
- we can retrieve the result, if it was previously computed

In practice, we can store

- either $\langle \vee, \{a, b\} : c\rangle$ in the Operation cache — boolean OR is commutative
- or $\langle \{a, b\} : c\rangle$ in the OR (or UNION) cache
- either $\langle \Rightarrow, a, b : c\rangle$ in the Operation cache — boolean IMPLIES is not commutative
- or $\langle a, b : c\rangle$ in the IMPLIES cache

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

We can store

- any set of markings $\mathcal{Y} \subseteq \widehat{\mathcal{X}} = \mathbb{B}^{|\mathcal{P}|}$ 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 $\mathcal{N}$, with a $2|\mathcal{P}|$-level BDD

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

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

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

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

Using BDDs for state space generation

An L-level BDD encodes a set of states $\mathcal{Y}$ as a subset of the potential state space $\widehat{\mathcal{X}} = \mathbb{B}^L$

$$\mathbf{i} \equiv (\mathbf{i}_L, ..., \mathbf{i}_1) \in \mathcal{Y} \;\Leftrightarrow\; \text{the corresponding path from the root leads to terminal } 1$$

A $2L$-level BDD encodes the next-state function $\mathcal{N} : \widehat{\mathcal{X}} \to 2^{\widehat{\mathcal{X}}}$

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

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

$$\mathcal{X}_{init} \qquad \mathcal{N}(\mathcal{X}_{init}) \qquad \mathcal{N}(\mathcal{N}(\mathcal{X}_{init})) \qquad \mathcal{N}(\mathcal{N}(\mathcal{N}(\mathcal{X}_{init}))) \qquad \cdots$$

The main operation is a repeated application of the relational product operator

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

$\mathcal{Y} \leftarrow \mathcal{X}_{init}$;	*known states*
$\mathcal{U} \leftarrow \mathcal{X}_{init}$;	*unexplored states*
while $\mathcal{U} \neq \emptyset$ do	
$\quad \mathcal{W} \leftarrow \mathcal{N}(\mathcal{U})$;	*potentially new states*
$\quad \mathcal{U} \leftarrow \mathcal{W} \setminus \mathcal{Y}$;	*truly new states*
$\quad \mathcal{Y} \leftarrow \mathcal{Y} \cup \mathcal{U}$;	
return $\mathcal{Y}$;	

sets and relations are encoded using BDDs

runtime is proportional to the BDD sizes

If $|\mathcal{X}_k| > 2$, use multiple boolean levels to encode $\mathbf{i}_k$

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 $n_k \in \mathbb{N}$

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

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 $\qquad \mathbf{0}.lvl = \mathbf{1}.lvl = 0$
- A nonterminal node p is at a level k, with $L \geq k \geq 1$ $\qquad p.lvl = k$
- 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 $\qquad p[i_k].lvl < p.lvl$
- 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:

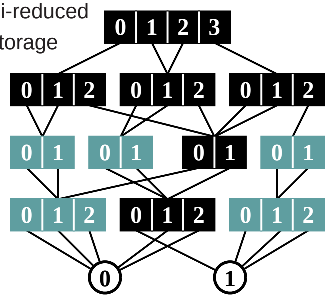
- The only root nodes with no incoming arcs are at level L
- If a node p is at level k, each child $p[i_k]$ is at level $k - 1$

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

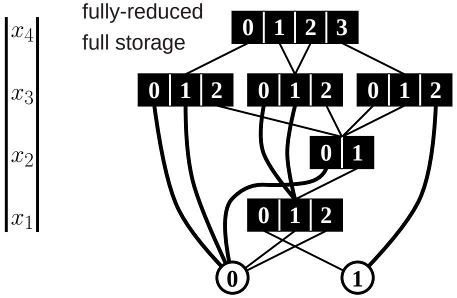
- There are no redundant nodes p at level k satisfying $p[i_k] = q$ for all $i_k \in \mathcal{X}_k$

Quasi-reduced vs. fully-reduced MDDs

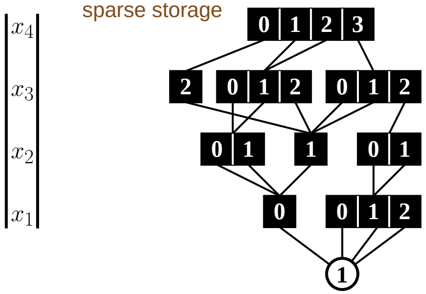
quasi-reduced
full storage



fully-reduced
full storage



sparse storage



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

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

All but a finite number of edges point to $\mathbf{0}$

Only nodes encoding $\emptyset$ are redundant

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

Locality

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

- can be modified by α: $\mathcal{V}_M(\alpha) = \{x_k : \exists \mathbf{i}, \mathbf{i}' \in \widehat{\mathcal{X}}, \mathbf{i}' \in \mathcal{N}_\alpha(\mathbf{i}) \wedge \mathbf{i}[k] \neq \mathbf{i}'[k]\}$
- can disable α: $\mathcal{V}_D(\alpha) = \{x_k : \exists \mathbf{i}, \mathbf{j} \in \widehat{\mathcal{X}}, \forall h \neq k, \mathbf{i}[h] = \mathbf{j}[h] \wedge \mathcal{N}_\alpha(\mathbf{i}) \neq \emptyset \wedge \mathcal{N}_\alpha(\mathbf{j}) = \emptyset\}$

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) $\{Top(\alpha), ..., Bot(\alpha)\}$ for event α is often much smaller than $\{L, ..., 1\}$

fully/quasi-reduced $2L$-level MDD encoding does not exploit locality

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

Kronecker description of the next-state function

$\mathcal{N} : \widehat{\mathcal{X}} \to 2^{\widehat{\mathcal{X}}}$ can be thought of as a boolean matrix $\mathbf{N} \in \mathbb{B}^{|\widehat{\mathcal{X}}| \times |\widehat{\mathcal{X}}|}$

The model is Kronecker-consistent if $\mathbf{N} = \sum_{\alpha \in \mathcal{E}} \left(\bigotimes_{L \geq k \geq 1} \mathbf{N}_{k,\alpha} \right)$ (using boolean operations)

In other words, $\mathcal{N} = \bigvee_{\alpha \in \mathcal{E}} \mathcal{N}_\alpha$ and each $\mathcal{N}_\alpha = \bigwedge_{L \geq k \geq 1} \mathcal{N}_{k,\alpha}$ where

$\mathcal{N}_{k,\alpha} : \mathcal{X}_k \to 2^{\mathcal{X}_k}$ is encoded by the boolean matrix $\mathbf{N}_{k,\alpha} \in \mathbb{B}^{|\mathcal{X}_k| \times |\mathcal{X}_k|}$

Locality: If the k^{th} local state does not affect and is not affected by event α, then $\mathbf{N}_{k,\alpha} = \mathbf{I}$

encode a huge $\mathbf{N}$ with $L \cdot |\mathcal{E}|$ "small" matrices

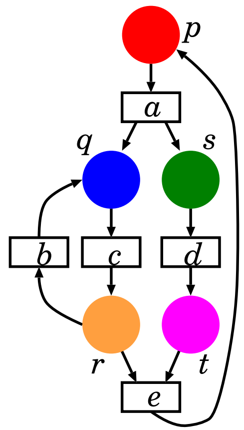
Using structural information to encode $\mathcal{N}$ $(L = 5)$

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

EVENTS →

LEVELS ↓

$\mathbf{N}_{5,a}$: ?	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{N}_{5,e}$: ?
$\mathbf{N}_{4,a}$: ?	$\mathbf{N}_{4,b}$: ?	$\mathbf{N}_{4,c}$: ?	$\mathbf{I}$	$\mathbf{I}$
$\mathbf{I}$	$\mathbf{N}_{3,b}$: ?	$\mathbf{N}_{3,c}$: ?	$\mathbf{I}$	$\mathbf{N}_{3,e}$: ?
$\mathbf{N}_{2,a}$: ?	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{N}_{2,d}$: ?	$\mathbf{I}$
$\mathbf{I}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{N}_{1,d}$: ?	$\mathbf{N}_{1,e}$: ?
$Top(a)$:5 $Bot(a)$:2	$Top(b)$:4 $Bot(b)$:3	$Top(c)$:4 $Bot(c)$:3	$Top(d)$:2 $Bot(d)$:1	$Top(e)$:5 $Bot(e)$:1



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

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

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

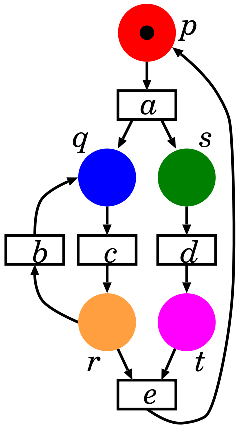
EVENTS →

LEVELS ↓

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

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

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



Using structural information to encode $\mathcal{N}$ $(L=4)$

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

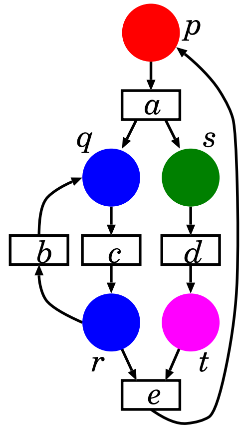
EVENTS →

LEVELS ↓

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

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

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



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

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

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

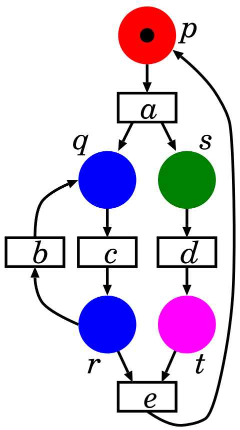
EVENTS →

LEVELS ↓

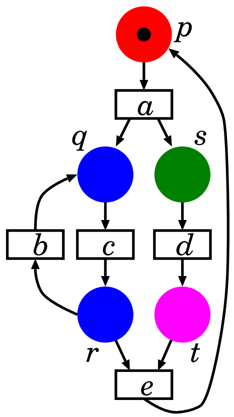
$\mathbf{N}_{4,a}$: $\begin{bmatrix}0&1\\0&0\end{bmatrix}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{N}_{4,e}$: $\begin{bmatrix}0&0\\1&0\end{bmatrix}$
$\mathbf{N}_{3,a}$: $\begin{bmatrix}0&1&0\\0&0&0\\0&0&0\end{bmatrix}$	$\mathbf{N}_{3,b}$: $\begin{bmatrix}0&0&0\\0&0&0\\0&1&0\end{bmatrix}$	$\mathbf{N}_{3,c}$: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

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

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



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

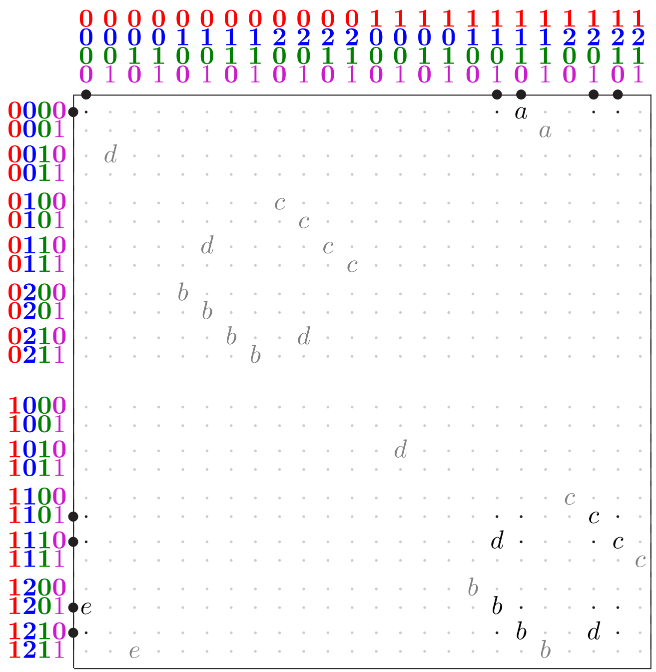


$\{p^1,p^0\}\equiv\{0,1\}$

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

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

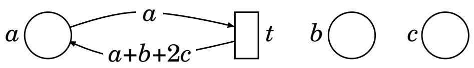
$\{t^0,t^1\}\equiv\{0,1\}$



Non-Kronecker-consistent models

To model software we must be able to model assignments, e.g., with self-modifying Petri nets

```
a:=a+b+2*c;
```



To enforce Kronecker consistency, places a, b, and c must belong to the same submodel

However, each assignment may cause further grouping of variables

`a := a+b+2*c;`	$\{a,b,c\}$
`...`	
`x := y+z;`	$\{a,b,c\}, \{x,y,z\}$
`...`	
`w := x+m;`	$\{a,b,c\}, \{x,y,z,w,m\}$
`...`	
`r := w+a;`	$\{a,b,c,x,y,z,w,m,r\}$

The local state space for $\{a,b,c,x,y,z,w,m,r\}$ may be too large

[Miner QEST 2004] [Ciardo and Yu CHARME 2005]

Self-modifying Petri nets with inhibitor arcs

A self-modifying Petri net with inhibitor arcs is a tuple $(\mathcal{P}, \mathcal{E}, \mathbf{D}^-, \mathbf{D}^+, \mathbf{D}^\circ, \mathbf{i}^{init})$ where:

- $\mathcal{P}$ and $\mathcal{E}$ — places and transitions
- $\mathbf{D}^-, \mathbf{D}^+ : \mathcal{P} \times \mathcal{E} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N}$ — marking-dependent input, output arc cardinalities
- $\mathbf{D}^\circ : \mathcal{P} \times \mathcal{E} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N} \cup \{\infty\}$ — marking-dependent inhibitor arc cardinalities
- $\mathbf{i}^{init} : \mathbb{N}^{|\mathcal{P}|}$ — initial marking

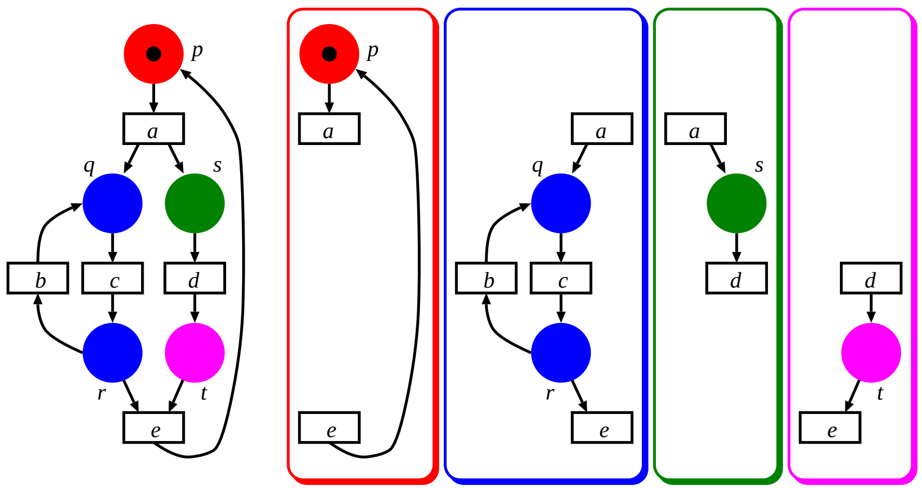
Transition α is enabled in marking $\mathbf{i} \in \mathbb{N}^{|\mathcal{P}|}$ iff $\quad \forall p \in \mathcal{P},\ \mathbf{D}^-_{p,\alpha}(\mathbf{i}) \le i_p \ \wedge\ \mathbf{D}^\circ_{p,\alpha}(\mathbf{i}) > i_p$

If α is enabled in $\mathbf{i}$, it can *fire* and lead to marking $\mathbf{j}$ $\quad \forall p \in \mathcal{P},\ \ j_p = i_p - \mathbf{D}^-_{p,\alpha}(\mathbf{i}) + \mathbf{D}^+_{p,\alpha}(\mathbf{i})$

The effect of α is deterministic, so we can write $\mathbf{i} \xrightarrow{\alpha} \mathbf{j}$ or use the general notation $\mathbf{j} \in \mathcal{N}_\alpha(\mathbf{i})$

$$\widehat{\mathcal{X}} \equiv \mathbb{N}^{|\mathcal{P}|} \qquad \mathcal{X}_{init} \equiv \{\mathbf{i}^{init}\} \qquad \mathcal{N} \equiv \bigcup_{\alpha \in \mathcal{E}} \mathcal{N}_\alpha$$

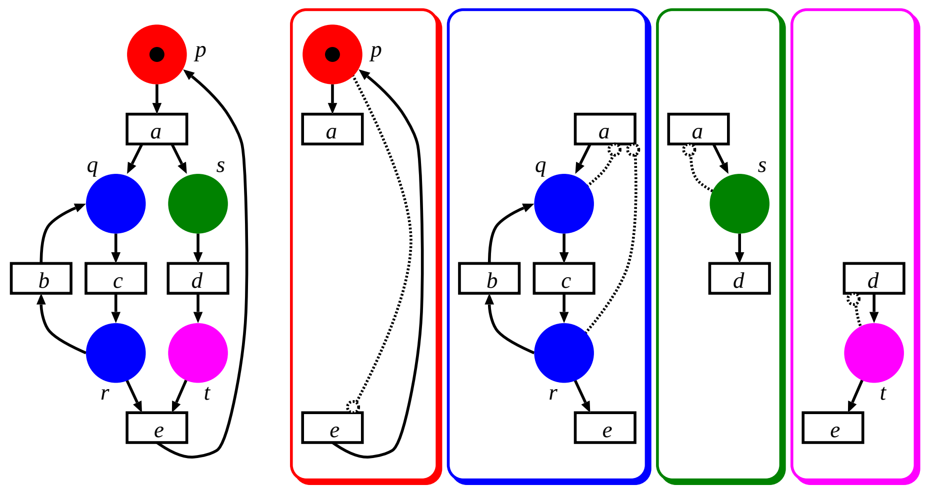
Problem with pre-generating the sets $\mathcal{X}_k$



each one of these four submodels is unbounded (in isolation)

A poor solution to the problem

Modifying the model to enforce known bounds is difficult (especially if we want the smallest $\mathcal{X}_k$)



more importantly, it's dangerous!

Locality in general structured models

Another way express Kronecker-consistency:

$\mathcal{N} : \widehat{\mathcal{X}} \to 2^{\widehat{\mathcal{X}}}$ tells us which transitions between potential states are possible

$\mathcal{N} = \bigvee_{\alpha \in \mathcal{E}} \mathcal{N}_\alpha, \quad \mathcal{N}_\alpha : \widehat{\mathcal{X}} \to 2^{\widehat{\mathcal{X}}}$ asynchronous, disjunctive, decomposition of $\mathcal{N}$

$\mathcal{N}_\alpha = (\bigwedge_{k \in \mathcal{D}_\alpha} \mathcal{N}_{k,\alpha}) \wedge (\bigwedge_{k \in \overline{\mathcal{D}_\alpha}} \mathcal{I}_k)$ synchronous, conjunctive, decomposition by level of $\mathcal{N}_\alpha$

$\mathcal{D}_\alpha \subseteq \{L, ..., 1\}$	set of levels on which event α depends/affects
$\overline{\mathcal{D}_\alpha} = \{L, ..., 1\} \setminus \mathcal{D}_\alpha$	set of levels on which event α does not depend/affect
$\mathcal{I}_k$	identity transformation for the states of submodel k
$\mathcal{N}_{k,\alpha} : \mathcal{X}_k \to 2^{\mathcal{X}_k}$	next-state function restricted to level k only

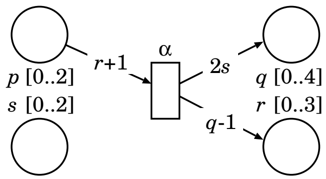
The disjunctive-then-conjunctive decomposition of $\mathcal{N}$ can be applied to arbirary models:

$\mathcal{N}_\alpha = (\bigwedge_{c=1}^{m_\alpha} \mathcal{N}_{\mathcal{D}_{c,\alpha},\alpha}) \wedge (\bigwedge_{k \in \overline{\mathcal{D}_\alpha}} \mathcal{I}_k)$ general conjunctive decomposition of $\mathcal{N}_\alpha$

$\bigcup_{c=1}^{m_\alpha} \mathcal{D}_{c,\alpha} = \mathcal{D}_\alpha \subseteq \{L, ..., 1\}$	$\mathcal{N}_{\mathcal{D}_{c,\alpha},\alpha}$ depends on a set of levels
$\mathcal{N}_{\mathcal{D}_{c,\alpha},\alpha} : (\times_{k \in \mathcal{D}_{c,\alpha}} \mathcal{X}_k) \to 2^{(\times_{k \in \mathcal{D}_{c,\alpha}} \mathcal{X}_k)}$	

$$Top(\alpha) = \max \mathcal{D}_\alpha \qquad Bot(\alpha) = \min \mathcal{D}_\alpha$$

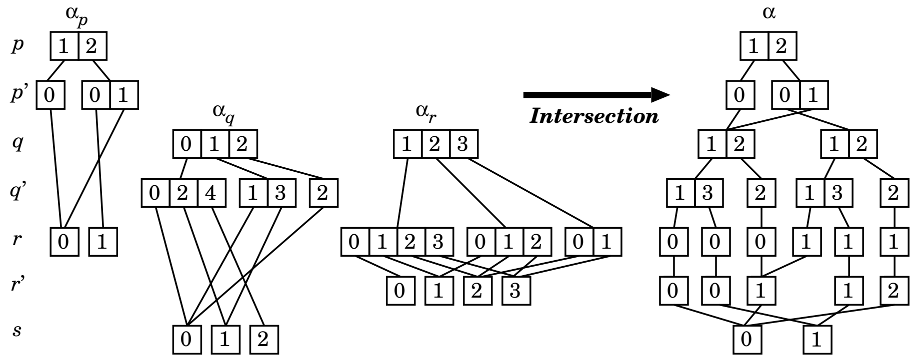
An example of general conjunctive decomposition



Equivalent pseudocode (simultaneous statements)

if $p > r \wedge q + 2s \leq 4 \wedge q > 0 \wedge r + q \leq 4$ then
$p \leftarrow p - (r + 1);$
$q \leftarrow q + 2s;$
$r \leftarrow r + q - 1;$

Assume $\mathcal{X}_p = \{0,1,2\}, \quad \mathcal{X}_q = \{0,1,2,3,4\}, \quad \mathcal{X}_r = \{0,1,2,3\}, \quad \mathcal{X}_s = \{0,1,2\}$



special reduction rule and interpretation of skipped levels

Building the MDDs encoding $\mathcal{N}$ on-the-fly

To confirm a new local state $\mathbf{i}_h \in \mathcal{X}_h$:

for $k = L$ down to h do
  for each α such that $Top(\alpha) = k$ and $h \in \mathcal{D}_\alpha$ do
    for each $\mathcal{D}_{c,\alpha}$ containing h do
      explicitly build the set $\mathcal{Y}$ of potential transitions from $\{\mathbf{i}_h\} \times (\times_{k \in \mathcal{D}_{c,\alpha} \setminus \{h\}} \mathcal{X}_k)$;
      $\mathcal{N}_{\mathcal{D}_{c,\alpha},\alpha} \leftarrow \mathcal{N}_{\mathcal{D}_{c,\alpha},\alpha} \cup \mathcal{Y}$; *build the conjunct*
    $\mathcal{N}_\alpha \leftarrow (\bigwedge_{c=1}^{m_\alpha} \mathcal{N}_{\mathcal{D}_{c,\alpha},\alpha}) \wedge (\bigwedge_{l \in \overline{\mathcal{D}_\alpha}} \mathcal{I}_l)$; *build the disjunct*
    $\mathcal{N}_k \leftarrow \mathcal{N}_k \cup \mathcal{N}_\alpha$; $\mathcal{N}_k = \bigcup_{Top(\alpha)=k} \mathcal{N}_\alpha$

the explicit enumeration of the elements of $\times_{h \in \mathcal{D}_{c,\alpha}} \mathcal{X}_h$
is the reason for seeking the smallest possible $\mathcal{D}_{c,\alpha}$

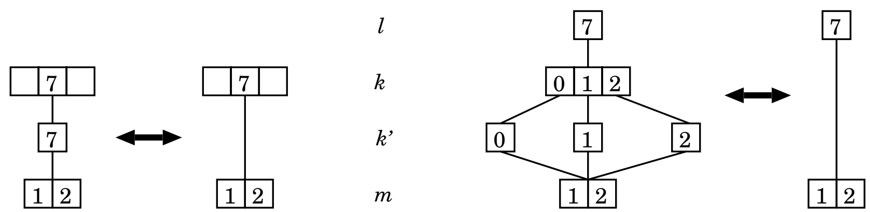
New reduction rule for MDDs

In addition to the

- quasi–reduced form
- reduced form

we need an

- identity–reduced form to be used for "to", or "primed", levels



Identity–reduced level k' Identity–reduced level k' and fully–reduced level k

canonical even if we use different reduction rules for each level

Accelerated fixpoint computation

Using MDDs to generate the state space $\mathcal{X}_{reach}$

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

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

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

$\mathbf{j} \in \mathcal{N}(\mathbf{i}) \;\Leftrightarrow\;$ 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

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

Standard method

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

- $\mathcal{Y} \leftarrow \mathcal{X}_{init}$; *known states*
- $\mathcal{U} \leftarrow \mathcal{X}_{init}$; *unexplored states*
- repeat
  - $\mathcal{W} \leftarrow \mathcal{N}(\mathcal{U})$; *potentially new states*
  - $\mathcal{U} \leftarrow \mathcal{W} \setminus \mathcal{Y}$; *truly new states*
  - $\mathcal{Y} \leftarrow \mathcal{Y} \cup \mathcal{U}$;
- until $\mathcal{U} = \emptyset$;
- return $\mathcal{Y}$;

Alternative All method

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

- $\mathcal{Y} \leftarrow \mathcal{X}_{init}$;
- repeat
  - $\mathcal{O} \leftarrow \mathcal{Y}$; *old states*
  - $\mathcal{Y} \leftarrow \mathcal{O} \cup \mathcal{N}(\mathcal{O})$; *new states*
- until $\mathcal{O} = \mathcal{Y}$;
- return $\mathcal{Y}$;

Chaining the next-state function $\mathcal{N}$ [Roig95]

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

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

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

- $\mathcal{Y} \leftarrow \mathcal{X}_{init}$; *known states*
- $\mathcal{U} \leftarrow \mathcal{X}_{init}$; *unexplored known states*
- repeat
  - $\mathcal{W} \leftarrow \emptyset$;
  - for each $\alpha \in \mathcal{E}$ do
    - $\mathcal{W} \leftarrow \mathcal{W} \cup \mathcal{N}_\alpha(\mathcal{U})$;
  - $\mathcal{U} \leftarrow \mathcal{W} \setminus \mathcal{Y}$; *truly new states*
  - $\mathcal{Y} \leftarrow \mathcal{Y} \cup \mathcal{U}$;
- until $\mathcal{U} = \emptyset$;
- return $\mathcal{Y}$;

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

- $\mathcal{Y} \leftarrow \mathcal{X}_{init}$; *known states*
- $\mathcal{U} \leftarrow \mathcal{X}_{init}$; *unexplored known states*
- repeat
  - for each $\alpha \in \mathcal{E}$ do
    - $\mathcal{U} \leftarrow \mathcal{U} \cup \mathcal{N}_\alpha(\mathcal{U})$;
    - $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathcal{Y}$; *truly new states*
  - $\mathcal{Y} \leftarrow \mathcal{Y} \cup \mathcal{U}$;
- until $\mathcal{U} = \emptyset$;
- return $\mathcal{Y}$;

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

- $\mathcal{Y} \leftarrow \mathcal{X}_{init}$; *known states*
- repeat
  - $\mathcal{O} \leftarrow \mathcal{Y}$; *save old state space*
  - $\mathcal{W} \leftarrow \emptyset$;
  - for each $\alpha \in \mathcal{E}$ do
    - $\mathcal{W} \leftarrow \mathcal{W} \cup \mathcal{N}_\alpha(\mathcal{O})$;
  - $\mathcal{Y} \leftarrow \mathcal{O} \cup \mathcal{W}$;
- until $\mathcal{O} = \mathcal{Y}$;
- return $\mathcal{Y}$;

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

- $\mathcal{Y} \leftarrow \mathcal{X}_{init}$; *known states*
- repeat
  - $\mathcal{O} \leftarrow \mathcal{Y}$; *save old state space*
  - for each $\alpha \in \mathcal{E}$ do
    - $\mathcal{Y} \leftarrow \mathcal{Y} \cup \mathcal{N}_\alpha(\mathcal{Y})$;
- until $\mathcal{O} = \mathcal{Y}$;
- return $\mathcal{Y}$;

Comparing the four approaches

N	$\lvert\mathcal{X}_{reach}\rvert$	Time (sec)				Memory (MB)				
		Bf	*AllBf*	*Ch*	*AllCh*	*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	<0.1
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: $L=N$, $\lvert\mathcal{X}_k\rvert=15$ **for all** k										
5	5.3×10^{4}	0.2	0.3	0.1	0.1	0.8	1.1	0.3	0.2	<0.1
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: $L=N+1$, $\lvert\mathcal{X}_k\rvert=10$ **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	<0.1
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: $L=19$, $\lvert\mathcal{X}_k\rvert=N+1$ **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	<0.1
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

Saturation: an iteration strategy based on the model structure

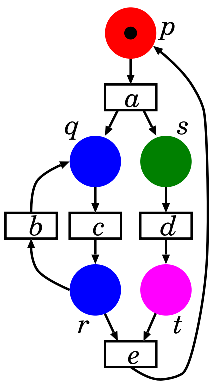
MDD node p at level k is ***saturated*** if it encodes a fixed point w.r.t. any event α s.t. the highest MDD level it depends on, $Top(\alpha)$, is at most k $\Rightarrow$ all MDD nodes reachable from p are also saturated

- build the L-level MDD encoding of $\mathcal{X}_{init}$ — if $\lvert\mathcal{X}_{init}\rvert=1$, 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 $L-1$, $L-2$, $\ldots$, 1, saturate them immediately upon creation)

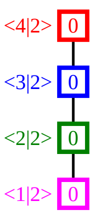
states are **not** discovered in breadth-first order

enormous time and memory savings for asynchronous systems

Saturation-OTF in action: initial setup



a	bc	d	e
	$\mathbf{I}$	$\mathbf{I}$	
		$\mathbf{I}$	
	$\mathbf{I}$		$\mathbf{I}$
$\mathbf{I}$	$\mathbf{I}$		



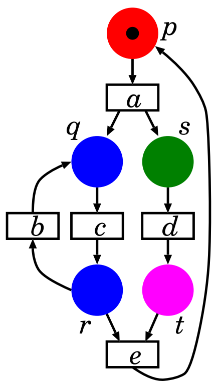
$\mathcal{X}_4=\{p^1\}\equiv\{0\}$

$\mathcal{X}_3=\{q^0r^0\}\equiv\{0\}$

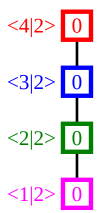
$\mathcal{X}_2=\{s^0\}\equiv\{0\}$

$\mathcal{X}_1=\{t^0\}\equiv\{0\}$

Saturation-OTF in action: confirm initial state



a	bc	d	e
$0:1$	$\mathbf{I}$	$\mathbf{I}$	$0:2$
$0:1$	$0:-$	$\mathbf{I}$	$0:-$
$0:1$	$\mathbf{I}$	$0:-$	$\mathbf{I}$
$\mathbf{I}$	$\mathbf{I}$	$0:1$	$0:-$



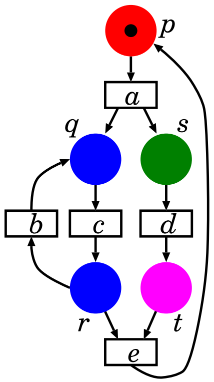
$\mathcal{X}_4=\{\underline{p^1},p^0,p^2\}\equiv\{\underline{0},1,2\}$

$\mathcal{X}_3=\{\underline{q^0r^0},q^1r^0\}\equiv\{\underline{0},1\}$

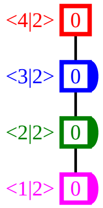
$\mathcal{X}_2=\{\underline{s^0},s^1\}\equiv\{\underline{0},1\}$

$\mathcal{X}_1=\{\underline{t^0},t^1\}\equiv\{\underline{0},1\}$

Saturation-OTF in action: saturate $\langle 1|2\rangle$, $\langle 2|2\rangle$, $\langle 3|2\rangle$ (no firing)



a	bc	d	e
0 : 1	I	I	0 : 2
0 : 1	0 : –	I	0 : –
0 : 1	I	0 : –	I
I	I	0 : 1	0 : –



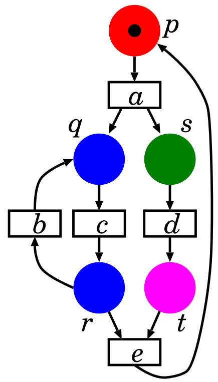
$\mathcal{X}_4=\{\underline{p^1},p^0,p^2\}\equiv\{\underline{0},1,2\}$

$\mathcal{X}_3=\{\underline{q^0r^0},q^1r^0\}\equiv\{\underline{0},1\}$

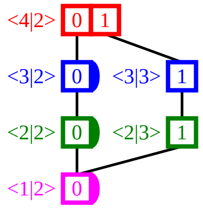
$\mathcal{X}_2=\{\underline{s^0},s^1\}\equiv\{\underline{0},1\}$

$\mathcal{X}_1=\{\underline{t^0},t^1\}\equiv\{\underline{0},1\}$

Saturation-OTF in action: saturate $\langle 4|2\rangle$ (fire a)



a	bc	d	e
0 : 1	I	I	0 : 2
0 : 1	0 : –	I	0 : –
0 : 1	I	0 : –	I
I	I	0 : 1	0 : –



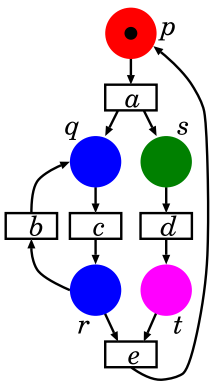
$\mathcal{X}_4=\{\underline{p^1},p^0,p^2\}\equiv\{\underline{0},1,2\}$

$\mathcal{X}_3=\{\underline{q^0r^0},q^1r^0\}\equiv\{\underline{0},1\}$

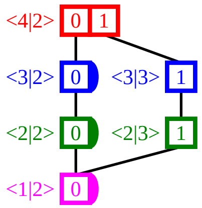
$\mathcal{X}_2=\{\underline{s^0},s^1\}\equiv\{\underline{0},1\}$

$\mathcal{X}_1=\{\underline{t^0},t^1\}\equiv\{\underline{0},1\}$

Saturation-OTF in action: confirm $s^1\equiv 1$



a	bc	d	e
0 : 1	I	I	0 : 2
0 : 1	0 : –	I	0 : –
0 : 1 1 : 2	I	0 : – 1 : 0	I
I	I	0 : 1	0 : –



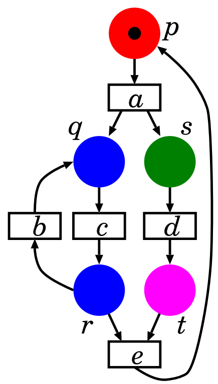
$\mathcal{X}_4=\{\underline{p^1},p^0,p^2\}\equiv\{\underline{0},1,2\}$

$\mathcal{X}_3=\{\underline{q^0r^0},q^1r^0\}\equiv\{\underline{0},1\}$

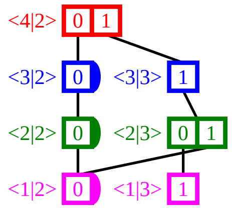
$\mathcal{X}_2=\{\underline{s^0},\underline{s^1},s^2\}\equiv\{\underline{0},\underline{1},2\}$

$\mathcal{X}_1=\{\underline{t^0},t^1\}\equiv\{\underline{0},1\}$

Saturation-OTF in action: saturate $\langle 2|3\rangle$ (fire d)



a	bc	d	e
0 : 1	I	I	0 : 2
0 : 1	0 : –	I	0 : –
0 : 1 1 : 2	I	0 : – 1 : 0	I
I	I	0 : 1	0 : –



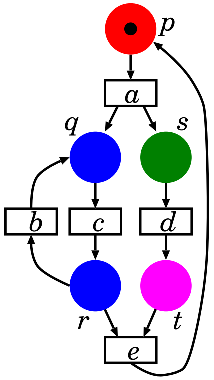
$\mathcal{X}_4=\{\underline{p^1},p^0,p^2\}\equiv\{\underline{0},1,2\}$

$\mathcal{X}_3=\{\underline{q^0r^0},q^1r^0\}\equiv\{\underline{0},1\}$

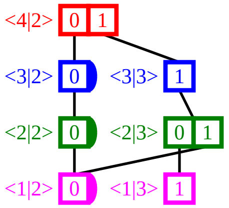
$\mathcal{X}_2=\{\underline{s^0},\underline{s^1},s^2\}\equiv\{\underline{0},\underline{1},2\}$

$\mathcal{X}_1=\{\underline{t^0},t^1\}\equiv\{\underline{0},1\}$

Saturation-OTF in action: confirm $t^1 \equiv 1$



a	bc	d	e
0 : 1	I	I	0 : 2
0 : 1	0 : –	I	0 : –
0 : 1 1 : 2	I	0 : – 1 : 0	I
I	I	0 : 1 1 : 2	0 : – 1 : 0



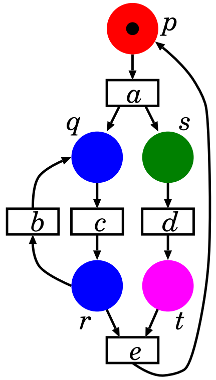
$\mathcal{X}_4 = \{\underline{p^1}, p^0, p^2\} \equiv \{\underline{0}, 1, 2\}$

$\mathcal{X}_3 = \{\underline{q^0r^0}, q^1r^0\} \equiv \{\underline{0}, 1\}$

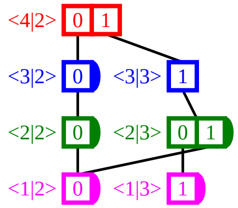
$\mathcal{X}_2 = \{\underline{s^0}, \underline{s^1}, s^2\} \equiv \{\underline{0}, \underline{1}, 2\}$

$\mathcal{X}_1 = \{\underline{t^0}, \underline{t^1}, t^2\} \equiv \{\underline{0}, \underline{1}, 2\}$

Saturation-OTF in action: saturate $\langle 1|3 \rangle$ (no firing)



a	bc	d	e
0 : 1	I	I	0 : 2
0 : 1	0 : –	I	0 : –
0 : 1 1 : 2	I	0 : – 1 : 0	I
I	I	0 : 1 1 : 2	0 : – 1 : 0



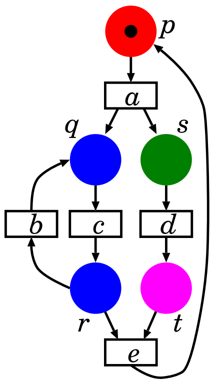
$\mathcal{X}_4 = \{\underline{p^1}, p^0, p^2\} \equiv \{\underline{0}, 1, 2\}$

$\mathcal{X}_3 = \{\underline{q^0r^0}, q^1r^0\} \equiv \{\underline{0}, 1\}$

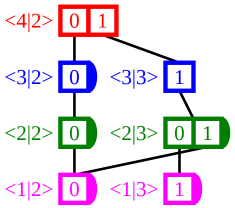
$\mathcal{X}_2 = \{\underline{s^0}, \underline{s^1}, s^2\} \equiv \{\underline{0}, \underline{1}, 2\}$

$\mathcal{X}_1 = \{\underline{t^0}, \underline{t^1}, t^2\} \equiv \{\underline{0}, \underline{1}, 2\}$

Saturation-OTF in action: confirm $q^1r^0 \equiv 1$



a	bc	d	e
0 : 1	I	I	0 : 2
0 : 1 1 : 2	0 : – 1 : 3	I	0 : – 1 : –
0 : 1 1 : 2	I	0 : – 1 : 0	I
I	I	0 : 1 1 : 2	0 : – 1 : 0



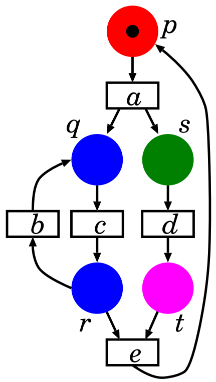
$\mathcal{X}_4 = \{\underline{p^1}, p^0, p^2\} \equiv \{\underline{0}, 1, 2\}$

$\mathcal{X}_3 = \{\underline{q^0r^0}, \underline{q^1r^0}, q^2r^0, q^0r^1\} \equiv \{\underline{0}, \underline{1}, 2, 3\}$

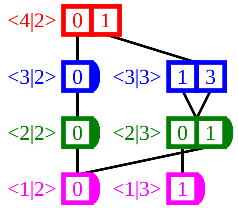
$\mathcal{X}_2 = \{\underline{s^0}, \underline{s^1}, s^2\} \equiv \{\underline{0}, \underline{1}, 2\}$

$\mathcal{X}_1 = \{\underline{t^0}, \underline{t^1}, t^2\} \equiv \{\underline{0}, \underline{1}, 2\}$

Saturation-OTF in action: saturate $\langle 3|3 \rangle$ (fire bc)



a	bc	d	e
0 : 1	I	I	0 : 2
0 : 1 1 : 2	0 : – 1 : 3	I	0 : – 1 : –
0 : 1 1 : 2	I	0 : – 1 : 0	I
I	I	0 : 1 1 : 2	0 : – 1 : 0



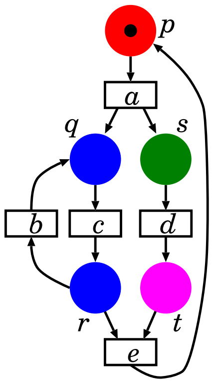
$\mathcal{X}_4 = \{\underline{p^1}, p^0, p^2\} \equiv \{\underline{0}, 1, 2\}$

$\mathcal{X}_3 = \{\underline{q^0r^0}, \underline{q^1r^0}, q^2r^0, q^0r^1\} \equiv \{\underline{0}, \underline{1}, 2, 3\}$

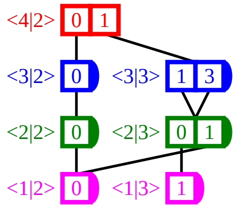
$\mathcal{X}_2 = \{\underline{s^0}, \underline{s^1}, s^2\} \equiv \{\underline{0}, \underline{1}, 2\}$

$\mathcal{X}_1 = \{\underline{t^0}, \underline{t^1}, t^2\} \equiv \{\underline{0}, \underline{1}, 2\}$

Saturation-OTF in action: confirm $q^0r^1 \equiv 3$



a	bc	d	e
0 : 1	I	I	0 : 2
0 : 1 1 : 2 3 : 4	0 : – 1 : 3 3 : 1	I	0 : – 1 : – 3 : 0
0 : 1 1 : 2	I	0 : – 1 : 0	I
I	I	0 : 1 1 : 2	0 : – 1 : 0



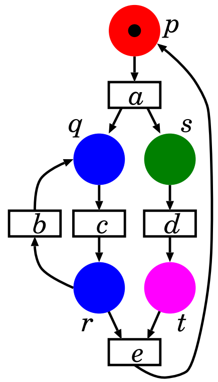
$\mathcal{X}_4 = \{\underline{p^1}, p^0, p^2\} \equiv \{\underline{0}, 1, 2\}$

$\mathcal{X}_3 = \{\underline{q^0r^0}, \underline{q^1r^0}, q^2r^0, \underline{q^0r^1}, q^1r^1\} \equiv \{\underline{0}, \underline{1}, 2, \underline{3}, 4\}$

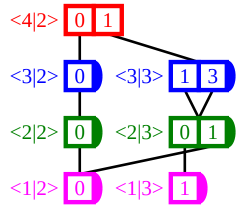
$\mathcal{X}_2 = \{\underline{s^0}, \underline{s^1}, s^2\} \equiv \{\underline{0}, \underline{1}, 2\}$

$\mathcal{X}_1 = \{\underline{t^0}, \underline{t^1}, t^2\} \equiv \{\underline{0}, \underline{1}, 2\}$

Saturation-OTF in action: confirm $p^0 \equiv 1$



a	bc	d	e
0 : 1 1 : –	I	I	0 : 2 1 : 0
0 : 1 1 : 2 3 : 4	0 : – 1 : 3 3 : 1	I	0 : – 1 : – 3 : 0
0 : 1 1 : 2	I	0 : – 1 : 0	I
I	I	0 : 1 1 : 2	0 : – 1 : 0



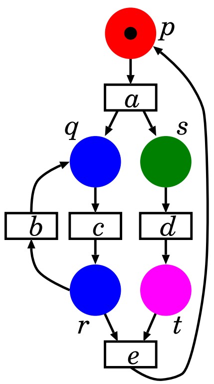
$\mathcal{X}_4 = \{\underline{p^1}, \underline{p^0}, p^2\} \equiv \{\underline{0}, \underline{1}, 2\}$

$\mathcal{X}_3 = \{\underline{q^0r^0}, \underline{q^1r^0}, q^2r^0, \underline{q^0r^1}, q^1r^1\} \equiv \{\underline{0}, \underline{1}, 2, \underline{3}, 4\}$

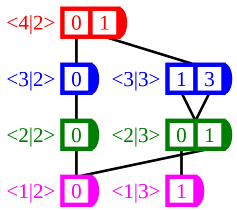
$\mathcal{X}_2 = \{\underline{s^0}, \underline{s^1}, s^2\} \equiv \{\underline{0}, \underline{1}, 2\}$

$\mathcal{X}_1 = \{\underline{t^0}, \underline{t^1}, t^2\} \equiv \{\underline{0}, \underline{1}, 2\}$

Saturation-OTF in action: saturate $\langle 4|2 \rangle$ (fire e)



a	bc	d	e
0 : 1 1 : –	I	I	0 : 2 1 : 0
0 : 1 1 : 2 3 : 4	0 : – 1 : 3 3 : 1	I	0 : – 1 : – 3 : 0
0 : 1 1 : 2	I	0 : – 1 : 0	I
I	I	0 : 1 1 : 2	0 : – 1 : 0



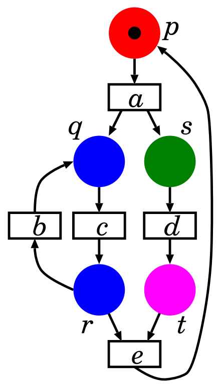
$\mathcal{X}_4 = \{\underline{p^1}, \underline{p^0}, p^2\} \equiv \{\underline{0}, \underline{1}, 2\}$

$\mathcal{X}_3 = \{\underline{q^0r^0}, \underline{q^1r^0}, q^2r^0, \underline{q^0r^1}, q^1r^1\} \equiv \{\underline{0}, \underline{1}, 2, \underline{3}, 4\}$

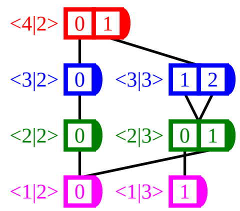
$\mathcal{X}_2 = \{\underline{s^0}, \underline{s^1}, s^2\} \equiv \{\underline{0}, \underline{1}, 2\}$

$\mathcal{X}_1 = \{\underline{t^0}, \underline{t^1}, t^2\} \equiv \{\underline{0}, \underline{1}, 2\}$

Saturation-OTF in action: remap confirmed indices



a	bc	d	e
0 : 1 1 : –	I	I	0 : – 1 : 0
0 : 1 1 : – 2 : –	0 : – 1 : 2 2 : 1	I	0 : – 1 : – 2 : 0
0 : 1 1 : –	I	0 : – 1 : 0	I
I	I	0 : 1 1 : –	0 : – 1 : 0



$\mathcal{X}_4 = \{\underline{p^1}, \underline{p^0}\} \equiv \{\underline{0}, \underline{1}\}$

$\mathcal{X}_3 = \{\underline{q^0r^0}, \underline{q^1r^0}, \underline{q^0r^1}\} \equiv \{\underline{0}, \underline{1}, \underline{2}\}$

$\mathcal{X}_2 = \{\underline{s^0}, \underline{s^1}\} \equiv \{\underline{0}, \underline{1}\}$

$\mathcal{X}_1 = \{\underline{t^0}, \underline{t^1}\} \equiv \{\underline{0}, \underline{1}\}$

Results (Saturation OTF vs. pregen vs. BFS)

N	Reachable states	Final memory (KB) OTF	Final memory (KB) PRE	Final memory (KB) NuSMV	Peak memory (KB) OTF	Peak memory (KB) PRE	Peak memory (KB) NuSMV	Time (sec) OTF	Time (sec) PRE	Time (sec) NuSMV
Dining Philosophers: $L=N/2$, $\lvert\mathcal{X}_k\rvert=34$ for all k										
20	3.46×10^{12}	4	3	4,178	5	4	4,192	0.01	0.01	0.4
50	2.23×10^{31}	11	10	8,847	14	12	8,863	0.03	0.02	13.1
100	4.97×10^{62}	24	20	8,891	28	25	15,256	0.06	0.05	990.8
200	2.47×10^{125}	**48**	**40**	**21,618**	**57**	**50**	**59,423**	**0.15**	**0.11**	**18,129.3**
5,000	6.53×10^{3134}	1,210	1,015	—	1,445	1,269	—	65.55	51.29	—
Slotted Ring Network: $L=N$, $\lvert\mathcal{X}_k\rvert=15$ for all k										
5	5.39×10^{4}	1	1	502	5	5	507	0.01	0.01	0.1
10	8.29×10^{9}	5	5	4,332	28	27	8,863	0.06	0.04	6.1
15	1.46×10^{15}	10	9	771	80	77	11,054	0.18	0.13	2,853.1
100	2.60×10^{105}	434	398	—	15,753	14,486	—	41.72	25.78	—
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	2.30×10^{4}	5	5	917	6	7	932	0.01	0.01	0.2
20	4.72×10^{7}	18	17	5,980	20	21	5,985	0.04	0.03	1.4
30	7.25×10^{10}	37	36	2,222	41	41	8,716	0.09	0.07	5.6
100	2.85×10^{32}	357	355	13,789	372	372	21,814	2.11	1.55	2,836.5
150	4.82×10^{47}	784	781	—	807	807	—	7.04	5.07	—
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	1.92×10^{4}	5	6	2,113	6	9	2,126	0.01	0.01	1.0
10	2.50×10^{9}	16	19	1,152	26	31	8,928	0.02	0.02	41.6
25	8.54×10^{13}	**86**	**135**	**17,045**	**163**	**239**	**152,253**	**0.16**	**0.11**	**17,321.9**
150	4.84×10^{23}	6,291	15,459	—	16,140	29,998	—	18.50	10.92	—

Saturation behavior and properties

Traditional approaches apply the global next-state function $\mathcal{N}$ 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 an 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 (finds more states)

usually enormous memory and time savings

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

Saturation pseudocode (for Kronecker-consistent models)

$Generate$(in **s** : array$[1..L]$ of lcl) : idx
$p \leftarrow \mathbf{1}$;
for $k=1$ to L do
  $r \leftarrow NewNode(k)$;
  $r[\mathbf{s}[k]] \leftarrow p$;
  $Saturate(k,r)$;
  $UniqueTableInsert(k,r)$;
  $p \leftarrow r$;
  return r;

$Saturate$(in $k : lvl, p : idx$)
repeat
  $pCng \leftarrow false$;
  foreach $\alpha \in \mathcal{E}_k$ do
    $\mathcal{L} \leftarrow Locals(k,\alpha,p)$;
    while $\mathcal{L} \neq \emptyset$ do
      $i \leftarrow Pick(\mathcal{L})$;
      $f \leftarrow RecFire(k-1,\alpha,p[i])$;
      if $f \neq \mathbf{0}$ then
        foreach $j \in \mathcal{N}_{k,\alpha}(i)$ do
          $u \leftarrow Union(k-1,f,p[j])$;
          if $u \neq p[j]$ then
            $p[j] \leftarrow u$;
            $pCng \leftarrow true$;
            if $\mathcal{N}_{k,\alpha}(j) \neq \emptyset$ then
              $\mathcal{L} \leftarrow \mathcal{L} \cup \{j\}$;
until $pCng = false$;

$RecFire$(in $h : lvl, \alpha : evnt, q : idx$) : idx
if $h < Bot(\alpha)$ then return q;
if $Find(FC[h],(q,\alpha),s)$ then return s;
$s \leftarrow NewNode(h)$;
$sCng \leftarrow false$;
$\mathcal{L} \leftarrow Locals(h,\alpha,q)$;
while $\mathcal{L} \neq \emptyset$ do
  $i \leftarrow Pick(\mathcal{L})$;
  $f \leftarrow RecFire(h-1,\alpha,q[i])$;
  if $f \neq \mathbf{0}$ then
    foreach $j \in \mathcal{N}_{h,\alpha}(i)$ do
      $u \leftarrow Union(h-1,f,s[j])$;
      if $u \neq s[j]$ then
        $s[j] \leftarrow u$;
        $sCng \leftarrow true$;
if $sCng$ then
  $Saturate(h,s)$;
$UniqueTableInsert(h,s)$;
$Insert(FC[h],(q,\alpha),s)$;
return s;

FC is the firing cache

Saturation pseudocode (for arbitrary models)

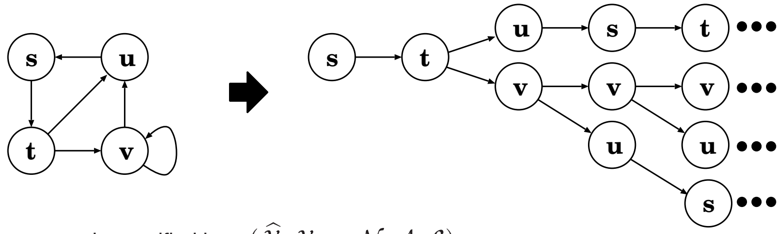
mdd $Saturate$(level k, mdd p) is — *quasi-reduced version*
local mdd $r, r_0, \ldots, r_{n_k-1}$;
if $k=0$ then return p;
if $Cache$ contains entry $\langle SaturateCODE, p : r\rangle$ then return r;
foreach $i_k \in \mathcal{X}_k$ do
  $r_{i_k} \leftarrow Saturate(k-1, p[i_k])$; — *first, be sure that the children are saturated*
repeat
  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 $\mathcal{N}_\alpha[i_k][j_k] \neq \mathbf{0}$;
  $r_{j_k} \leftarrow Union(r_{j_k}, RelProdSat(k-1, r_{i_k}, \mathcal{N}_\alpha[i_k][j_k]))$;
until $r_0, \ldots, r_{n_k-1}$ do not change;
$r \leftarrow UniqueTableInsert(k, r_0, \ldots, r_{n_k-1})$;
enter $\langle SaturateCODE, p : r\rangle$ in $Cache$;
return r;

mdd $RelProdSat$(level k, mdd q, mdd2 f) is
local mdd $r, r_0, \ldots, r_{n_k-1}$;
if $k=0$ then return $q \wedge f$;
if $Cache$ contains entry $\langle RelProdSatCODE, q, f : r\rangle$ then return r;
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
  $r_{j_k} \leftarrow Union(r_{j_k}, RelProdSat(k-1, q[i_k], f[i_k][j_k]))$;
$r \leftarrow Saturate(k, UniqueTableInsert(k, r_0, \ldots, r_{n_k-1}))$;
enter $\langle RelProdSatCODE, q, f : r\rangle$ in $Cache$;
return r.

CTL model checking of Petri nets

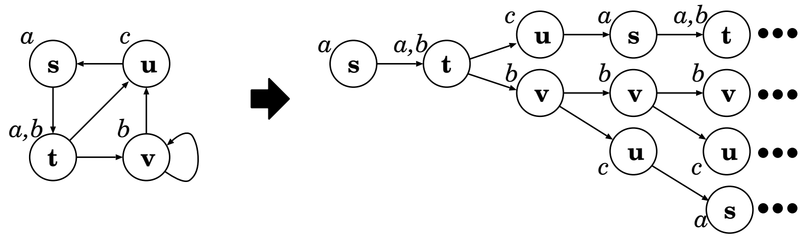
Kripke structures and computation trees

Given $(\widehat{\mathcal{X}}, \mathcal{X}_{init}, \mathcal{N})$, we can unwind the graph into a computation treee rooted at each $\mathbf{i} \in \mathcal{X}_{init}$



A Kripke structure is specified by $(\widehat{\mathcal{X}}, \mathcal{X}_{init}, \mathcal{N}, \mathcal{A}, \mathcal{L})$

- $\mathcal{A}$ is a set of atomic properties
- $\mathcal{L} : \widehat{\mathcal{X}} \rightarrow 2^{\mathcal{A}}$ is a labeling function listing the atomic properties that hold in each state



CTL: computation tree logic

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

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

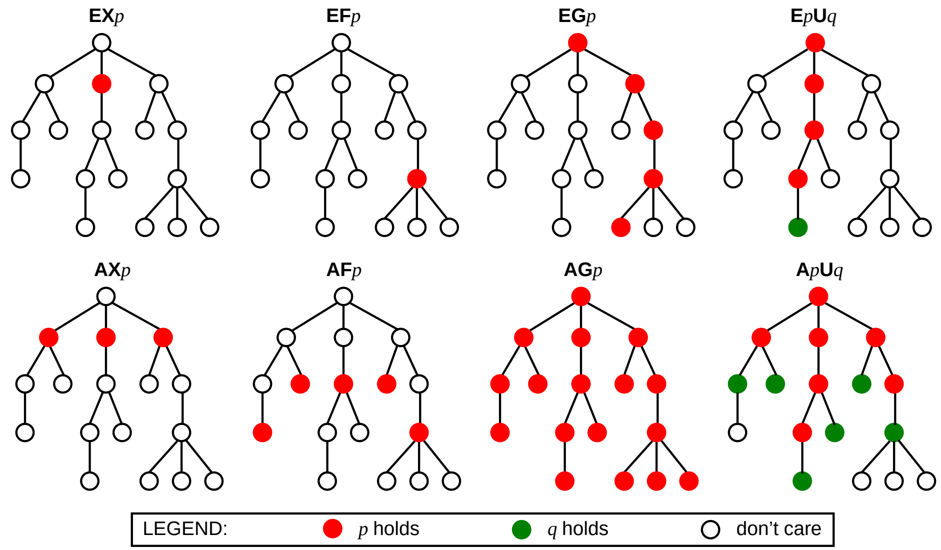
In CTL, operators occur in pairs:

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

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

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:

- AXp = ¬EX¬p
- EFp = E$true$Up
- AGp = ¬EF¬p
- EpRq = ¬A¬pU¬q
- AFp = ¬EG¬p
- ApUq = ¬(E¬qU¬p ∧ ¬q) ∧ ¬EG¬q
- ApRq = ¬E¬pU¬q

Examples of CTL statements

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

- $initial \Rightarrow$ AF (AG $stable$) or $initial \Rightarrow$ AF (A $stable$ U $shutdown$)

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

- AG EF $reset$

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

- AG $request\_critical \Rightarrow$ AF $access\_critical$

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

$LabelEX(p)$ is
$\mathcal{Y} \leftarrow \{\mathbf{i} \in \mathcal{X}_{reach} : p \in labels(\mathbf{i})\}$; *initialize $\mathcal{Y}$ with the states satisfying p*
while $\mathcal{Y} \neq \emptyset$ do
  pick and remove a state $\mathbf{j}$ from $\mathcal{Y}$;
  for each $\mathbf{i} \in \mathcal{N}^{-1}(\mathbf{j})$ do *state $\mathbf{i}$ can transition to state $\mathbf{j}$*
    $labels(\mathbf{i}) \leftarrow labels(\mathbf{i}) \cup \{\text{EX}p\}$;

If we have $\mathcal{X}_{reach} \subseteq \widehat{\mathcal{X}}$, we can assume $\mathcal{N} : \mathcal{X}_{reach} \rightarrow 2^{\mathcal{X}_{reach}}$ instead of $\mathcal{N} : \widehat{\mathcal{X}} \rightarrow 2^{\widehat{\mathcal{X}}}$

The EU algorithm for CTL (explicit version)

An algorithm to label all states that satisfy EpUq

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

$LabelEU(p, q)$ is
$\mathcal{Y} \leftarrow \{\mathbf{i} \in \mathcal{X}_{reach} : q \in labels(\mathbf{i})\}$; *initialize $\mathcal{Y}$ with the states satisfying q*
for each $\mathbf{i} \in \mathcal{Y}$ do
  $labels(\mathbf{i}) \leftarrow labels(\mathbf{i}) \cup \{\text{E}p\text{U}q\}$;
while $\mathcal{Y} \neq \emptyset$ do
  pick and remove a state $\mathbf{j}$ from $\mathcal{Y}$;
  for each $\mathbf{i} \in \mathcal{N}^{-1}(\mathbf{j})$ do *state $\mathbf{i}$ can transition to state $\mathbf{j}$*
    if EpU$q \notin labels(\mathbf{i})$ and $p \in labels(\mathbf{i})$ then
      $labels(\mathbf{i}) \leftarrow labels(\mathbf{i}) \cup \{\text{E}p\text{U}q\}$;
      $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{\mathbf{i}\}$;

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

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

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

The EX algorithm for CTL (symbolic version)

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

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

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

$BuildEXsymbolic(\mathcal{P})$ is
return $RelationalProduct(\mathcal{P}, \mathcal{N}^{-1})$; *perform one backward step in the transition relation*

Where

- $\mathcal{N}^{-1}$ is the inverse or backwards transition relation:

$$\mathbf{i} \in \mathcal{N}^{-1}(\mathbf{j}) \quad \Leftrightarrow \quad \mathbf{j} \in \mathcal{N}(\mathbf{i})$$

- given a relation $\mathcal{R} : \mathcal{A} \to 2^{\mathcal{B}}$ and a set $\mathcal{Y} \subseteq \mathcal{A}$:

$$RelationalProduct(\mathcal{Y}, \mathcal{R}) = \mathcal{R}(\mathcal{Y}) = \bigcup_{\mathbf{i} \in \mathcal{Y}} \mathcal{R}(\mathbf{i}) \subseteq \mathcal{B}$$

The EU algorithm for CTL (symbolic version)

Two algorithms to build the DD encoding the set of states that satisfy EpUq
Assume that the DDs encoding the sets $\mathcal{P}$ and $\mathcal{Q}$ of states satisfying p and q have been built already

$BuildEUsymbolic(\mathcal{P}, \mathcal{Q})$ is
$\mathcal{Y} \leftarrow \emptyset$;
$\mathcal{U} \leftarrow \mathcal{Q}$; *initialize the unexplored set $\mathcal{U}$ with the states satisfying q*
repeat
  $\mathcal{Y} \leftarrow Union(\mathcal{Y}, \mathcal{U})$; *currently known states satisfying EpUq*
  $\mathcal{W} \leftarrow RelationalProduct(\mathcal{U}, \mathcal{N}^{-1})$; *perform one backward step in the transition relation*
  $\mathcal{Z} \leftarrow Intersection(\mathcal{W}, \mathcal{P})$; *discard the states that do not satisfy p*
  $\mathcal{U} \leftarrow Difference(\mathcal{Z}, \mathcal{Y})$; *discard the states that are not new*
until $\mathcal{U} = \emptyset$;
return $\mathcal{Y}$;

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

The EG algorithm for CTL (symbolic version)

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

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

$BuildEGsymbolic(\mathcal{P})$ is
$\mathcal{Y} \leftarrow \mathcal{P}$; *initialize $\mathcal{Y}$ with the states satisfying p*
repeat
  $\mathcal{O} \leftarrow \mathcal{Y}$; *save the old set of states*
  $\mathcal{W} \leftarrow RelationalProduct(\mathcal{Y}, \mathcal{N}^{-1})$; *perform one backward step in the transition relation*
  $\mathcal{Y} \leftarrow Intersection(\mathcal{Y}, \mathcal{W})$;
until $\mathcal{O} = \mathcal{Y}$;
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 $\mathcal{N}$

Structural symbolic CTL model checking

Traditional symbolic CTL model checking (EF, EU, EG) uses a breadth-first fixed-point iteration

Just like for state-space generation, breadth-first can require huge peak memory, hence runtime

Using the model structure results in better algorithms for symbolic CTL model checking:

- exploit locality
- employ a Saturation-based algorithm for EF and EU
- greatly reduced memory and time requirements for asynchronous systems
- implemented in our tool SMART
- ○ can we apply Saturation to EG?
- ○ can we extend this to fair CTL?

Substantial time and memory improvements for EX and EG (thanks to locality)

Enormous time and memory improvements for EF and EU (thanks to locality and saturation)

Model checking results: SMART vs. NuSMV

$\lvert\mathcal{S}\rvert$	NuSMV				SMART						NuSMV				SMART	
(depends on	after SS		alone		$EUtrad$			$EUsat$			after SS		alone		$EGtrad$	
parameter N)	sec	kB	sec	kB	iter	sec	kB	iter	sec	kB	sec	kB	sec	kB	sec	kB
Phils					E$[(phil_1 \neq eat)$ U $(phil_0 = eat)]$						EG$(phil_0 \neq eat)$				starvation	
2.23×10^{31}	1.2	46	39.7	46	100	0.17	1	4	0.06	1	0.9	46	132.3	50	0.02	1
4.96×10^{62}	7.9	316	1121.8	316	200	0.67	3	4	0.14	3	9.0	316	2525.3	358	0.05	3
3.03×10^{313}	—	—	—	—	1000	19.09	78	4	0.77	60	—	—	—	—	0.28	58
FMS					E$[(M_1 > 0)$ U $(P_1s = P_2s = P_3s = N)]$						EG$\neg(P_1s = P_2s = P_3s = N)$					
3.44×10^{3}	0.2	17	318.1	43	31	0.04	$<.5$	6	0.01	$<.5$	0.2	17	128.9	18	$<.005$	$<.5$
4.86×10^{4}	1.0	127	—	—	46	0.16	$<.5$	8	0.02	$<.5$	1.0	127	—	—	0.01	$<.5$
8.54×10^{13}	—	—	—	—	376	—	—	52	1010.85	293	—	—	—	—	50.38	251
Round robin					E$[(p_1 \neq load)$ U $(p_0 = send)]$						EG$(true)$				find all cycles	
2.30×10^{5}	0.2	11	85.0	11	39	0.01	$<.5$	11	0.01	$<.5$	0.3	11	78.5	13	$<.005$	$<.5$
1.10×10^{6}	0.6	40	4922.7	40	59	0.03	$<.5$	16	0.01	$<.5$	1.2	40	4739.5	44	0.01	$<.5$
2.85×10^{32}	—	—	—	—	399	13.32	32	101	4.67	19	—	—	—	—	1.29	20
Leader					E$[(pref_1 = 0)$ U $(status_0 = leader)]$						EG$(status_0 \neq leader)$					
1.15×10^{4}	2.3	11	8104.7	371	62	0.36	1	38	0.27	1	232.8	12	1189.1	235	0.11	2
1.50×10^{5}	52.0	33	—	—	81	3.74	7	52	3.09	7	18023.6	104	—	—	0.44	9
2.39×10^{7}	—	—	—	—	121	690.85	116	85	416.85	101	—	—	—	—	7.15	128
Slotted ring					E$[(slot_1 \neq bf)$ U $(slot_0 = ag)]$						EG$(slot_0 \neq hg)$					
8.29×10^{9}	0.2	10	0.4	3	63	0.01	$<.5$	9	0.01	$<.5$	0.6	10	0.1	1	0.01	$<.5$
1.46×10^{15}	1.8	15	2.0	10	93	0.37	1	9	0.02	$<.5$	4.7	15	0.2	2	0.01	$<.5$
3.03×10^{105}	—	—	—	—	603	—	—	9	1.60	62	—	—	—	—	0.62	62

Decision diagrams for integer-valued functions

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 $n_0 \in \mathbb{N}$ (any set $\mathcal{X}_0$ will actually do)

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 $\quad \forall i_0 \in \mathcal{X}_0, i_0.lvl = 0$
- A nonterminal node p is at a level k, with $L \geq k \geq 1$ $\quad p.lvl = k$
- 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; $\quad p[0].lvl < p.lvl, p[1].lvl < p.lvl$
- 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 $p = q$

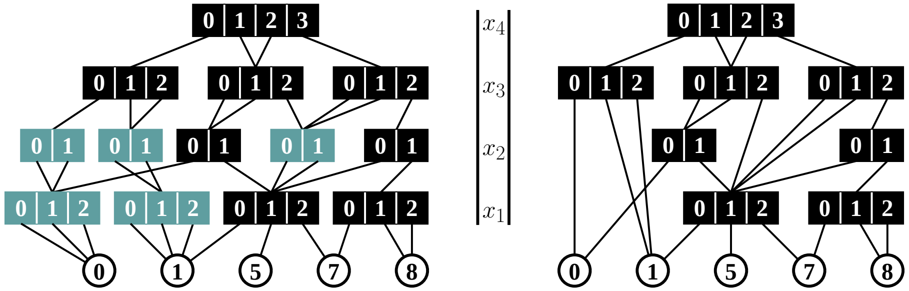
Then, if the MTMDD is quasi-reduced, there is no level skipping:

- 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



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

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

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

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

These MTMDDs encode a function $\widehat{\mathcal{X}} \to \mathbb{N}$ (or $\mathbb{Z}$, or $\mathbb{R}$, or any arbitrary set)

The distance function

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

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

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

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

```
d ← 0;
X_reach ← X_init;
X^[0] ← X_init;
repeat
    X^[d+1] ← N(X^[d]) \ X_reach;
    d ← d + 1;
    X_reach ← X_reach ∪ X^[d];
until X^[d] = ∅;
```

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

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

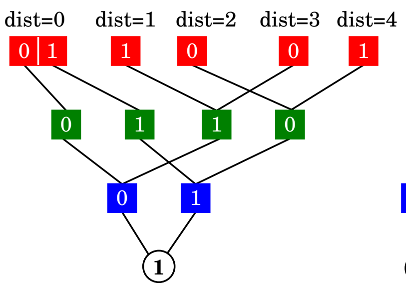
```
d ← 0;
Y^[0] ← X_init;
repeat
    Y^[d+1] ← N(Y^[d]) ∪ Y^[d];
    d ← d + 1;
until Y^[d] = Y^[d-1];
```

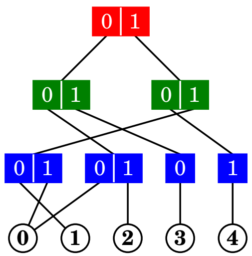
This is breadth-first symbolic state space generation

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

Encoding the distance function: MDD vs. MTMDD

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

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 $\mathbb{Z}$ and the combinator "$+$" (addition over the integers)

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 $L \ge k \ge 1$ — $p.lvl = k$
- 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 $v_{\langle \sigma, p \rangle} : \widehat{\mathcal{X}} \to \mathbb{Z}$ defined recursively by

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

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 $p[j_k].val = 0$ 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:

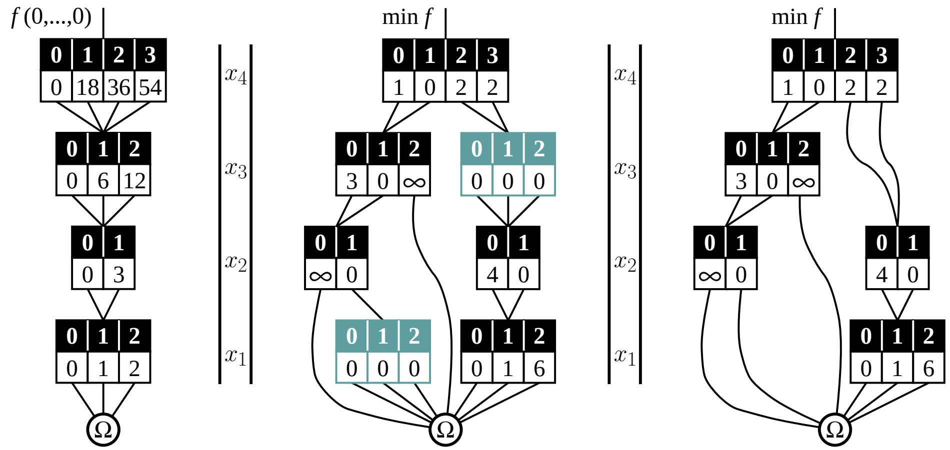
- There are no redundant nodes p satisfying $p[i_k].child = q$ and $p[i_k].val = 0$ for all $i_k \in \mathcal{X}_k$

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

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

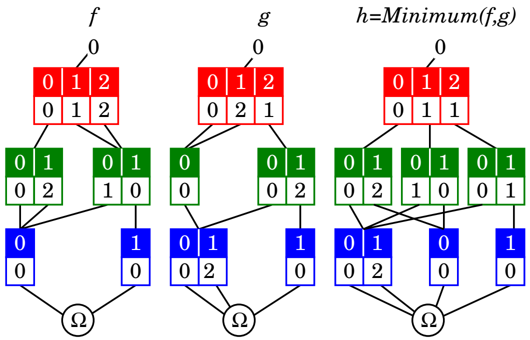
For EV$^+$MDDs, the value of the incoming root edge is $\min f$

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



Example: the *Minimum* operator for quasi-reduced EV$^+$MDDs

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



The *Minimum* operator for quasi-reduced EV$^+$MDDs


```
edge Minimum(level k, edge ⟨α,p⟩, edge ⟨β,q⟩)          edge is a pair ⟨int,node⟩
local node  p′, q′, r;
local int  μ, α′, β′;
local local  i_k;
if α = ∞ then return ⟨β,q⟩;
if β = ∞ then return ⟨α,p⟩;
μ ← min{α, β};
if k = 0 then return ⟨μ,Ω⟩;                            the only node at level 0 is Ω
if Cache contains entry ⟨MinimumCODE, k, p, q, α − β : γ, r⟩ then return ⟨γ + μ,r⟩;
r ← NewNode(k);                     create new node at level k with edges set to ⟨∞,Ω⟩
foreach i_k ∈ X_k do
    p′ ← p.child[i_k];
    α′ ← α − μ + p.val[i_k];
    q′ ← q.child[i_k];
    β′ ← β − μ + q.val[i_k];
    r[i_k] ← Minimum(k−1, ⟨α′,p′⟩, ⟨β′,q′⟩);                       continue downstream
UniqueTableInsert(k, r);
enter ⟨MinimumCODE, k, p, q, α − β : μ, r⟩ in Cache;
return ⟨μ,r⟩;
```

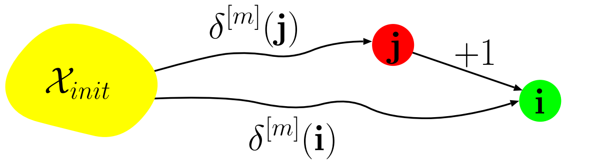
The distance function revisited

The distance function δ is 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

$$\delta^{[0]}(\mathbf{i}) = \begin{cases} 0 & \text{if } \mathbf{i} \in \mathcal{X}_{init} \\ \infty & \text{otherwise} \end{cases}$$

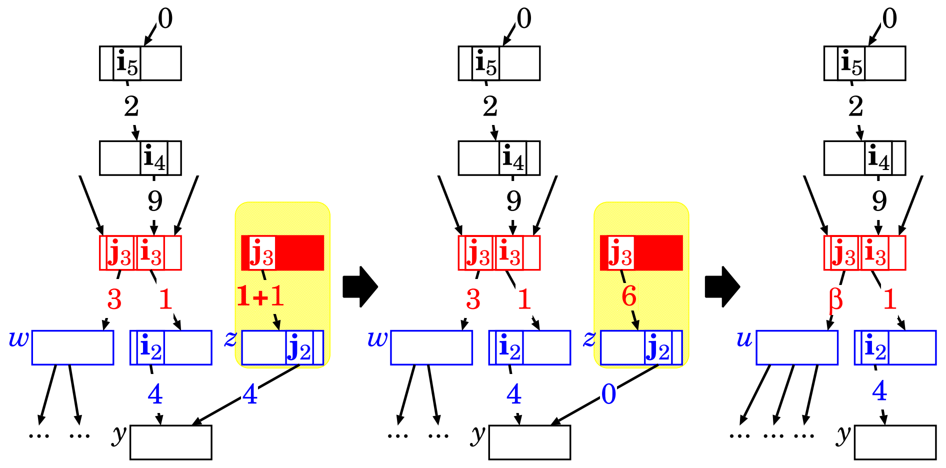


- At each iteration, we monotonically reduce the value of $\delta^{[m]}(\mathbf{i})$ for at least one state $\mathbf{i}$
- The traditional breadth-first iteration does this in an all-or-nothing fashion:

  $\delta^{[m]}(\mathbf{i}) = \infty$ for $m < d$, the actual distance d of $\mathbf{i}$, then $\delta^{[m]}(\mathbf{i}) = d$ for $m \geq d$
- The Saturation approach may instead reduce $\delta^{[m]}(\mathbf{i})$ multiple times, until it reaches d

Event firing in the EV$^+$MDD Saturation algorithm

effect of event e : $\mathbf{i}_3 \xrightarrow{e} \mathbf{j}_3$, $\mathbf{i}_2 \xrightarrow{e} \mathbf{j}_2$ (synchronously)



where $(\beta, u) = Min(\ (6, z),\ (3, w)\ )$

Results: time and memory to generate and store δ

		Time (in seconds)					Final nodes			Peak nodes				
N	$\lvert\mathcal{S}\rvert$	E_s	E_b	M_b	T_s	T_b	E_sE_b	M_b	T_sT_b	E_s	E_b	M_b	T_s	T_b
Dining philosophers: $d_{max}=2N$, $L=N/2$, $\lvert\mathcal{X}_k\rvert=34$ 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: $d_{max}=14N$, $L=4$, $\lvert\mathcal{X}_k\rvert=(N+3)(N+2)(N+1)/6$ 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.: $d_{max}=14N$, $L=19$, $\lvert\mathcal{X}_k\rvert=N+1$ for all k except $\lvert\mathcal{X}_{17}\rvert=4$, $\lvert\mathcal{X}_{12}\rvert=3$, $\lvert\mathcal{X}_2\rvert=2$														
5	2.9×10^{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: $d_{max}=8N-6$, $L=N+1$, $\lvert\mathcal{X}_k\rvert=10$ for all k except $\lvert\mathcal{X}_1\rvert=N+1$														
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

Generating an EF trace using EV$^+$MDDs

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 $\mathbf{j}^{[0]}, \ldots, \mathbf{j}^{[\mu]}$ 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 $\mathbf{i} \in \widehat{\mathcal{X}} \setminus \mathcal{X}$
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 $\mathbf{j}^{[\mu]} = (j_L^{[\mu]}, \ldots, j_1^{[\mu]})$ 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 $\mathbf{j}^{[\nu]} \in \mathcal{N}(\mathbf{i})$ (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 $\mathbf{i}^*$

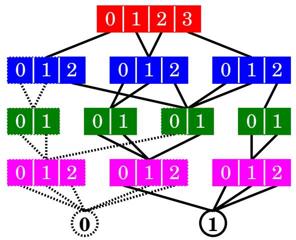
EV$^+$MDDs to store the lexicographic state indexing function ψ

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

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

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

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



$$\mathcal{Y} = \left\{ \begin{smallmatrix}0\\2\\1\\0\end{smallmatrix}, \begin{smallmatrix}1\\0\\0\\0\end{smallmatrix}, \begin{smallmatrix}1\\0\\1\\0\end{smallmatrix}, \begin{smallmatrix}1\\1\\0\\0\end{smallmatrix}, \begin{smallmatrix}1\\1\\1\\0\end{smallmatrix}, \begin{smallmatrix}1\\2\\1\\0\end{smallmatrix}, \begin{smallmatrix}2\\0\\0\\0\end{smallmatrix}, \begin{smallmatrix}2\\0\\1\\0\end{smallmatrix}, \begin{smallmatrix}2\\1\\0\\0\end{smallmatrix}, \begin{smallmatrix}2\\1\\1\\0\end{smallmatrix}, \begin{smallmatrix}2\\2\\1\\0\end{smallmatrix}, \begin{smallmatrix}3\\0\\1\\0\end{smallmatrix}, \begin{smallmatrix}3\\1\\1\\0\end{smallmatrix}, \begin{smallmatrix}3\\2\\0\\0\end{smallmatrix}, \begin{smallmatrix}3\\2\\0\\1\end{smallmatrix}, \begin{smallmatrix}3\\2\\0\\2\end{smallmatrix}, \begin{smallmatrix}3\\2\\1\\0\end{smallmatrix}, \begin{smallmatrix}3\\2\\1\\1\end{smallmatrix}, \begin{smallmatrix}3\\2\\1\\2\end{smallmatrix} \right\}$$

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

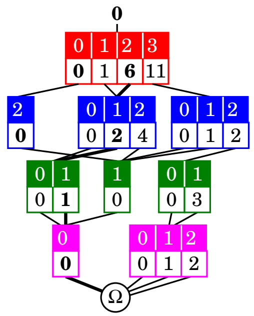
- Sum the values found on the corresponding path:

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

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

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

(a missing edge has the default value of ∞)



Numerical analysis of stochastic Petri nets

Continuous-time 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: $\mathbf{h}[\mathbf{i}] = 1 / \sum_{\mathbf{j} \in \mathcal{X}_{reach}} \mathbf{R}[\mathbf{i},\mathbf{j}]$
- $\boldsymbol{\pi}(0)[\mathbf{i}] = \Pr\{$chain is in state $\mathbf{i}$ at time 0, 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 $\dfrac{d\boldsymbol{\pi}(t)}{dt} = \boldsymbol{\pi}(t) \cdot \mathbf{Q}$ with initial condition $\boldsymbol{\pi}(0)$

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

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

Explicit generation of $\mathcal{X}_{reach}$ and $\mathbf{R}$

Explore(in: $\mathcal{X}_{init}, \mathcal{N}$; out: $\mathcal{X}_{reach}, \mathbf{R}, \psi$) is

$n \leftarrow 0$; — *state indices start at 0*
$\mathcal{X}_{reach} \leftarrow \emptyset$; — *$\mathcal{X}_{reach}$ contains the states explored so far*
$\mathcal{U} \leftarrow \mathcal{X}_{init}$; — *$\mathcal{U}$ contains the unexplored states known so far*
for each $\mathbf{i} \in \mathcal{X}_{init}$ do
  $\psi(\mathbf{i}) \leftarrow n$++; — *assign to $\mathbf{i}$ the next available index and increment n*
end for
while $\mathcal{U} \neq \emptyset$ do
  choose a state $\mathbf{i}$ in $\mathcal{U}$ and move it from $\mathcal{U}$ to $\mathcal{X}_{reach}$;
  for each event $\alpha \in \mathcal{E}$ and each state $\mathbf{j} \in \mathcal{N}_\alpha(\mathbf{i})$ do
    if $\mathbf{j} \notin \mathcal{X}_{reach} \cup \mathcal{U}$ then — *search to determine whether $\mathbf{j}$ is a new state*
      $\psi(\mathbf{j}) \leftarrow n$++; — *assign to $\mathbf{j}$ the next available index and increment n*
      $\mathcal{U} \leftarrow \mathcal{U} \cup \{\mathbf{j}\}$; — *remember to explore $\mathbf{j}$ later*
    end if;
    $\mathbf{R}[\psi(\mathbf{i}), \psi(\mathbf{j})] \leftarrow \mathbf{R}[\psi(\mathbf{i}), \psi(\mathbf{j})] + \lambda_\alpha(\mathbf{i})\Delta_\alpha(\mathbf{i},\mathbf{j})$; — *$\psi$ is used to index $\mathbf{R}$*
  end for;
end while;

$\psi : \widehat{\mathcal{X}} \to \{0, ..., |\mathcal{X}_{reach}|-1\} \cup \{\text{null}\}$ is a state indexing function (e.g., discovery order)

$\lambda_\alpha(\mathbf{i})$ is the rate at which event α fires in state $\mathbf{i}$

$\Delta_\alpha(\mathbf{i},\mathbf{j})$ is the probability that, if event α fires in state $\mathbf{i}$, the next state is $\mathbf{j}$

Kronecker-consistent decomposition of a CTMC model

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

- the potential transition rate matrix $\widehat{\mathbf{R}}$ is additively partitioned $\boxed{\widehat{\mathbf{R}} = \sum_{\alpha\in\mathcal{E}} \widehat{\mathbf{R}}_\alpha}$
- $\widehat{\mathcal{S}} = \mathcal{X}_L \times \cdots \times \mathcal{X}_1$, a global state $\mathbf{i}$ consists of L local states $\boxed{\mathbf{i} = (\mathbf{i}_L, \ldots, \mathbf{i}_1)}$
- and, most importantly, we can multiplicatively partition each $\widehat{\mathbf{R}}_\alpha$, that is, we can write

$\boxed{\lambda_\alpha(\mathbf{i}) = \lambda_{L,\alpha}(\mathbf{i}_L) \cdots \lambda_{1,\alpha}(\mathbf{i}_1)}$ and $\boxed{\Delta_\alpha(\mathbf{i},\mathbf{j}) = \Delta_{L,\alpha}(\mathbf{i}_L,\mathbf{j}_L) \cdots \Delta_{1,\alpha}(\mathbf{i}_1,\mathbf{j}_1)}$

$$\widehat{\mathbf{R}}_\alpha = \mathbf{R}_{L,\alpha} \otimes \cdots \otimes \mathbf{R}_{1,\alpha}$$

We encode the potential transition rate matrix $\widehat{\mathbf{R}}$ with $|\mathcal{E}| \times L$ small matrices $\mathbf{R}_{k,\alpha} \in \mathbb{R}^{n_k \times n_k}$

for stochastic Petri nets with transition rates depending on at most one place, any partition of the places into L subsets is consistent (even with inhibitor, reset, or probabilistic arcs)

Generalized stochastic Petri nets

A GSPN is a tuple $(\mathcal{P}, \mathcal{E}_T, \mathcal{E}_V, \mathbf{D}^-, \mathbf{D}^+, \mathbf{D}^\circ, G, \succ, \mathbf{x}_{init}, \lambda, w)$ where:

- $\mathcal{E} = \mathcal{E}_T \cup \mathcal{E}_V$ is a set of transitions, or events
- $(\mathcal{P}, \mathcal{E}, \mathbf{D}^-, \mathbf{D}^+, \mathbf{D}^\circ, G, \succ, \mathbf{x}_{init})$ is a self-modifying PN with inibitor arcs, guards, priorities
- $\lambda : \mathcal{E}_T \times \mathbb{N}^{|P|} \to [0, +\infty)$ are state-dependent firing rates
- $w : \mathcal{E}_V \times \mathbb{N}^{|P|} \to [0, +\infty)$ are state-dependent firing weights

Events in $\mathcal{E}_T$ are timed, events in $\mathcal{E}_V$ are immediate

We require that $\lambda_\alpha(\mathbf{i}) = 0 \;\Leftrightarrow\; \alpha$ is not (state-) enabled in $\mathbf{i}$

We require that $w_\alpha(\mathbf{i}) = 0 \;\Leftrightarrow\; \alpha$ is not enabled in $\mathbf{i}$

The firing distribution of event α in state $\mathbf{i}$ is

- Undefined if α is not enabled in $\mathbf{i}$
- Expo$(\lambda_\alpha(\mathbf{i}))$ if $\alpha \in \mathcal{E}_T(\mathbf{i})$
- Const(0) if $\alpha \in \mathcal{E}_V(\mathbf{i})$

Enabling rule for GSPNs

Partition the reachability set $\mathcal{X}_{reach}$ into

- vanishing states (drawn with dotted lines): $\mathcal{V} = \{\mathbf{i} : \exists\alpha \in \mathcal{E}_V, \alpha \text{ is enabled in } \mathbf{i}\}$
- tangible states (drawn with solid lines): $\mathcal{T} = \mathcal{X}_{reach} \setminus \mathcal{V} = \{\mathbf{i} : \forall\alpha \in \mathcal{E}_V, \alpha \text{ is disabled in } \mathbf{i}\}$

If a state $\mathbf{i}$ enables an immediate event, no timed event can fire in $\mathbf{i}$

All timed events are (stochastically-) disabled in the vanishing states

The expected holding time in tangible state $\mathbf{i}$ is:
$$\mathbf{h}[\mathbf{i}] = \frac{1}{\sum_{\alpha\in\mathcal{E}_T(\mathbf{i})} \lambda_\alpha(\mathbf{i})}$$

The firing probability of enabled event $\alpha \in \mathcal{E}_T$ in tangible state $\mathbf{i}$ is:
$$\widehat{w}_\alpha(\mathbf{i}) = \lambda_\alpha(\mathbf{i}) \cdot \mathbf{h}[\mathbf{i}]$$

The firing probability of enabled event $\alpha \in \mathcal{E}_V$ in vanishing state $\mathbf{i}$ is:
$$\widehat{w}_\alpha(\mathbf{i}) = \frac{w_\alpha(\mathbf{i})}{\sum_{\beta\in\mathcal{E}_V(\mathbf{i})} w_\beta(\mathbf{i})}$$

The stochastic processes defined by a GSPN

Consider the stochastic process $\{(\mathbf{i}^{[n]}, \alpha^{[n]}, t^{[n]}) : n \in \mathbb{N}\}$ where

- $\mathbf{i}^{[n]}$ is the n-th state entered by the GSPN — $\mathbf{i}^{[0]} = \mathbf{x}_{init}$
- $\alpha^{[n]}$ is the n-th event to fire — $\alpha^{[0]}$ is undefined
- $t^{[n]}$ is the time of the n-th firing — $t^{[0]} = 0$

$\{\mathbf{i}^{[n]} : n \in \mathbb{N}\}$ is a discrete-time Markov chain (DTMC) over $\mathcal{X}_{reach} = \mathcal{T} \cup \mathcal{V}$

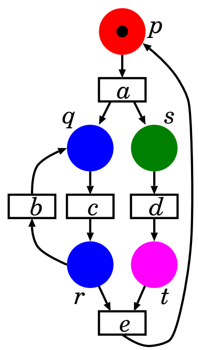
Consider sojourns into tangible states:

- $\mathbf{i}(\theta) = \mathbf{i} \ \Leftrightarrow\ \mathbf{i} = \mathbf{i}^{[n]}$ and $\theta^{[n]} \leq \theta < \theta^{[n+1]}$

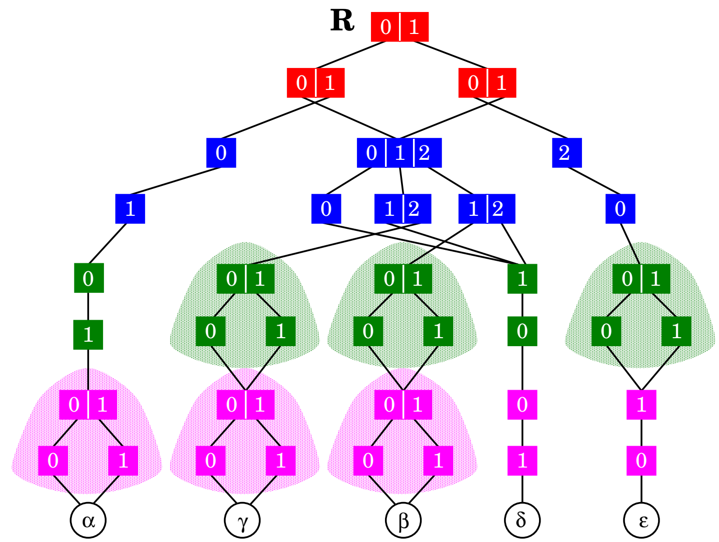
$\{\mathbf{i}(\theta) : \theta \geq 0\}$ is a continuous-time Markov chain (CTMC) over $\mathcal{X}_{reach} = \mathcal{T}$

A real-valued MTMDD: the transition rate matrix of an SPN

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



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



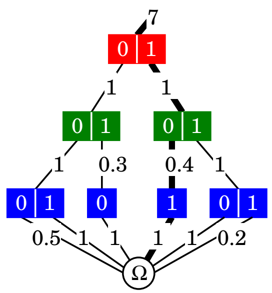
note the shaded identity patterns!!!

EV*MDDs by example

One way to think about EV*MDDs is "EV⁺MDD = $-\log($EV*MDD$)$":

0	$\Leftrightarrow$	1
edge values $\in [0, +\infty]$	$\Leftrightarrow$	edge values $\in [0, 1]$
root incoming edge $\in (-\infty, +\infty]$	$\Leftrightarrow$	root incoming edge $\in [0, +\infty)$
values add along the path	$\Leftrightarrow$	values multiply along the path

$\mathbf{i}_3$	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	3.5	7	2.1	0	0	2.8	7	1.4

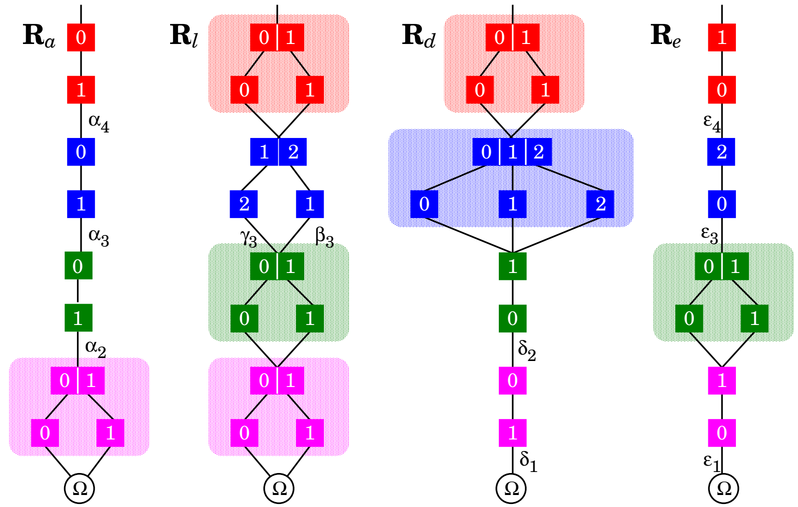


Canonicity: all edge values are in $[0, 1]$ and at least one is 1

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

Encoding $\mathbf{R}$ with an EV*MDD: initial non-canonical EV*MDDs

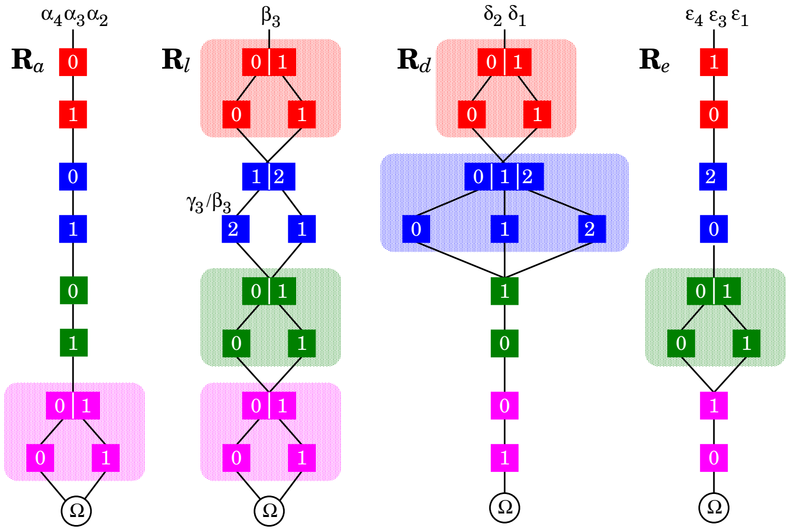
We can store $\mathbf{R}$ with a $2K$-level EV*MDD: consider the example of Kronecker encoding
$\mathbf{R} = \sum_{t\in\{a,l,d,e\}} \mathbf{R}_t = \sum_{t\in\{a,l,d,e\}} \bigotimes_{4\geq k\geq 1} \mathbf{R}_{k,t}$



note the shaded identity patterns!!!

Encoding $\mathbf{R}$ with an EV*MDD: canonical EV*MDDs

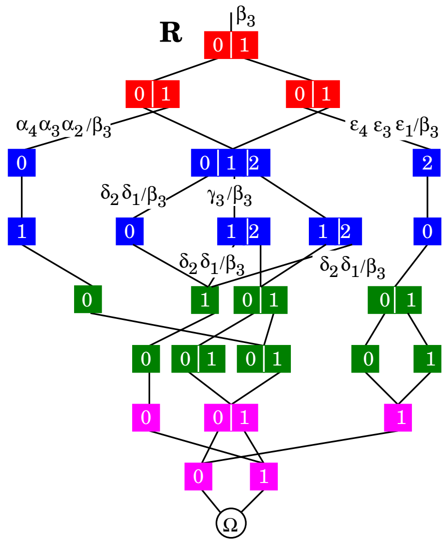
(assume that β_3 is the largest rate in $\mathbf{R}$)



Encoding $\mathbf{R}$ with an EV*MDD: the resulting canonical EV*MDD

Use a recursive algorithm to compute

$\mathbf{R} = \sum_{t\in\{a,l,d,e\}} \mathbf{R}_t$



hidden identity patterns remain!!!

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 $\mathbb{R}^{\geq 0} = [0, +\infty)$ and the combinator "$\cdot$" (multiplication over the reals)

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

- The only terminal node is Ω and is at level 0 $\quad \Omega.lvl = 0$
- 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 $n_k \times n_k$ 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 $\quad p[i_k, i'_k].child.lvl < p.lvl$
- An edge $\langle \sigma, p \rangle$, with $p.lvl = k$ encodes the function $v_{\langle \sigma,p \rangle} : \widehat{\mathcal{X}} \to \mathbb{Z}$ defined recursively by

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

This definition of $v_{\langle\sigma,p\rangle}$ applies when no edge skips a level, otherwise we have more choices...

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:

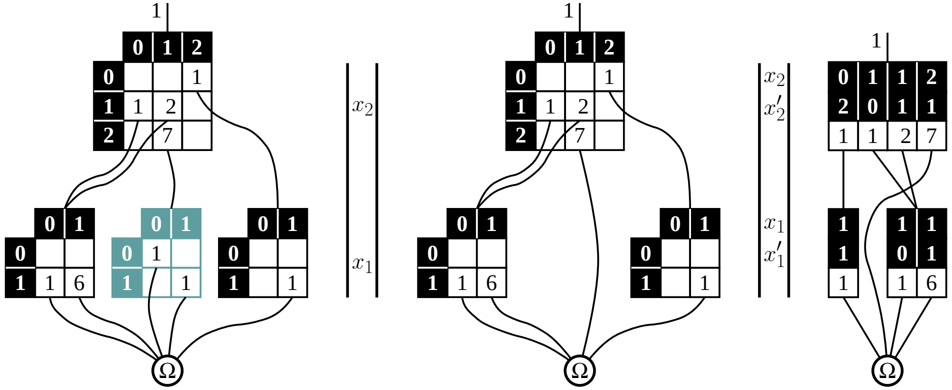
- $p[i_k, i_k].child = q$ and $p[i_k, i_k].val = 1$ for all $i_k \in \mathcal{X}_k$
- $p[i_k, i'_k].val = 0$ for all $i_k \neq i'_k$

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

Rows and columns of the matrix are indexed by $x_2 \cdot 2 + x_1$ and $x'_2 \cdot 2 + x'_1$

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

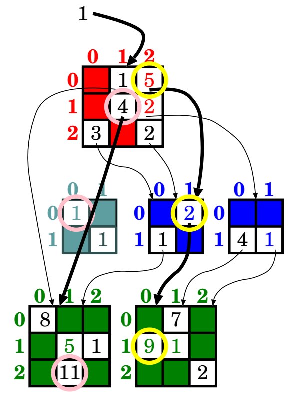
	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	1
2	0	0	0	0	0	0
3	1	6	2	12	0	0
4	0	0	7	0	0	0
5	0	0	0	7	0	0

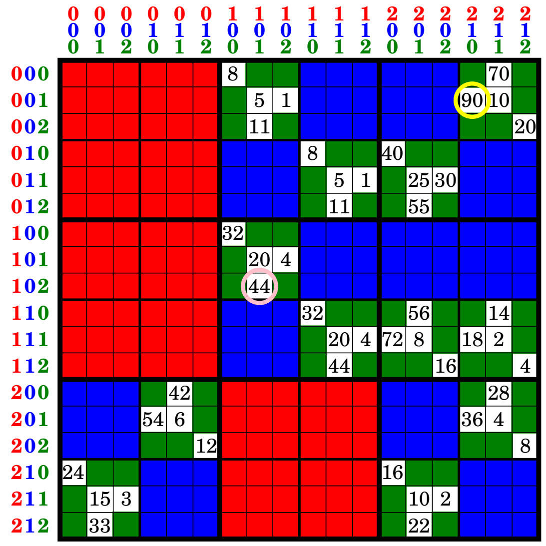


An example of MxD

R[001,210] = 1*5*2*9 = 90

R[102,101] = 1*4*11 = 44





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

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

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

- We can restrict $\mathcal{N}$ to $\mathcal{N} : \mathcal{X}_{reach} \to 2^{\mathcal{X}_{reach}}$ (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 — *$n = |\mathcal{X}_{reach}|$*
local natural s; — *state index in $\mathbf{x}$*
local real$[n]$ $\mathbf{y}$;
local sparse_real $\mathbf{c}$;
$s \leftarrow 0$;
for each $\mathbf{j} = (j_L, ..., j_1) \in \mathcal{X}_{reach}$ in lexicographic order do — *$s = \psi(\mathbf{j})$*
  $\mathbf{c} \leftarrow GetCol(L, A, \psi, j_L, ..., j_1)$; — *build column $\mathbf{j}$ of A using sparse storage*
  $\mathbf{y}[s] \leftarrow ElementWiseMult(\mathbf{x}, \mathbf{c})$; — *$\mathbf{x}$ uses full storage, $\mathbf{c}$ uses sparse storage*
  $s \leftarrow s + 1$;
return $\mathbf{y}$;

sparse_real $GetCol$(level k, mxd_node M, evmdd_node ϕ, natural $j_k, ..., j_1$) is
local sparse_real $\mathbf{c}, \mathbf{d}$;
if $k = 0$ then return $[1]$; — *a vector of size one, with its entry set to 1*
if $Cache$ contains entry $\langle GetColCODE, M, \phi, j_k, ..., j_1 : \mathbf{c}\rangle$ then return $\mathbf{c}$;
$\mathbf{c} \leftarrow \mathbf{0}$; — *initialize the result to all zero entries*
for each $i_k \in \mathcal{X}_k$ such that $M[i_k, j_k].val \neq 0$ and $\phi[i_k].val \neq \infty$ do
  $\mathbf{d} \leftarrow GetCol(k-1, M[i_k, j_k].child, \phi[i_k].child, j_{k-1}, ..., j_1)$;
  for each i such that $\mathbf{d}[i] \neq 0$ do
    $\mathbf{c}[i + \phi[i_k].val] \leftarrow \mathbf{c}[i + \phi[i_k].val] + M[i_k, j_k].val \cdot \mathbf{d}[i]$;
enter $\langle GetColCODE, M, \phi, j_k, ..., j_1 : \mathbf{c}\rangle$ in $Cache$;
return $\mathbf{c}$;

Empirical comparison

Memory consumption in bytes for:

$\mathcal{X}_{reach}$ (MDD), $\mathbf{R}$ (Sparse), $\widehat{\mathbf{R}}$ (Kronecker), $\widehat{\mathbf{R}}$ and $\mathbf{R}$ (Pot/Act MxD), $\widehat{\mathbf{R}}$ and $\mathbf{R}$ (Pot/Act MTMDD)

Model	N	$\lvert\widehat{\mathcal{X}}\rvert$	$\lvert\mathcal{X}_{reach}\rvert$	MDD	Sparse	Kron	Pot MxD	Act MxD	Pot MTMDD	Act MTMDD
qn4	2	324	324	333	14,256	772	586	722	22,784	22,784
	6	38,416	38,416	499	2,524,480	3,092	2,494	2,870	36,864	36,864
	10	527,076	527,076	905	38,524,464	7,076	5,778	6,522	62,720	62,720
qn8	2	6,561	324	681	14,256	1,204	738	1,688	43,776	49,152
	6	5,764,801	38,416	1,119	2,524,480	2,404	1,674	5,872	55,040	70,912
	10	214,358,881	527,076	1,953	38,524,464	3,604	2,610	12,040	66,304	98,560
mserv2	3	1,485	495	705	23,352	4,124	3,246	3,952	34,560	40,704
	6	6,345	2,115	3,176	111,408	17,468	13,998	16,432	111,104	135,168
	10	18,495	6,165	8,846	342,720	52,228	42,278	49,032	306,560	378,460
mserv4	3	14,256	495	1,174	23,352	5,568	4,098	4,916	68,864	79,616
	6	106,596	2,115	8,453	111,408	22,920	17,502	20,054	254,360	298,856
	10	488,268	6,165	33,739	342,720	67,560	52,342	58,934	873,896	998,552
mserv6	3	32,076	495	1,333	23,352	5,724	4,066	5,316	86,784	101,376
	6	239,841	2,115	8,614	111,408	23,076	17,470	20,238	298,596	347,956
	10	1,098,603	6,165	33,900	342,720	67,716	52,310	59,118	982,396	1,112,684

Model	N	$\lvert\widehat{\mathcal{X}}\rvert$	$\lvert\mathcal{X}_{reach}\rvert$	MDD	Sparse	Kron	Pot MxD	Act MxD	Pot MTMDD	Act MTMDD
molloy4	5	4,536	91	660	4,204	1,316	1,148	2,534	23,552	28,160
	8	32,805	285	1,215	14,676	2,528	2,300	5,216	27,648	38,656
	10	87,846	506	1,766	27,104	3,556	3,288	7,504	31,232	47,360
molloy5	5	7,776	91	846	4,204	1,100	792	4,298	28,416	37,120
	8	59,049	285	1,545	14,676	1,592	1,188	9,356	31,232	50,944
	10	161,051	506	2,223	27,104	1,920	1,452	13,778	33,280	61,952
kan3	1	160	160	264	8,032	500	412	544	18,432	18,432
	3	58,400	58,400	937	5,590,400	7,572	6,786	8,134	66,816	67,072
	5	2,546,432	2,546,432	5,646	303,705,920	45,660	41,816	48,780	303,776	303,776
kan4	1	256	160	332	8,032	420	354	602	23,552	24,576
	3	160,000	58,400	628	5,590,400	2,500	2,216	3,284	44,032	50,176
	5	9,834,496	2,546,432	1,532	303,705,920	7,940	7,118	9,950	92,928	110,592
kan16	1	65,536	160	1,275	8,032	2,148	866	3,000	95,232	107,520
	3	—	58,400	1,902	5,590,400	3,236	1,746	10,566	115,456	151,808
	5	—	2,546,432	3,149	303,705,920	4,324	2,626	24,106	135,168	216,320
fms5	1	2,100	84	535	3,228	1,456	604	1,808	36,096	40,960
	3	9,432,500	20,600	3,294	1,554,080	8,304	5,224	24,320	151,296	247,040
	5	2,016,379,008	852,012	30,490	82,727,748	34,484	24,664	138,244	654,892	1,255,108
fms21	1	4,194,304	84	2,050	3,228	3,132	1,132	7,396	126,976	148,224
	3	—	20,600	6,777	1,554,080	5,028	2,328	68,762	176,896	437,760
	5	—	852,012	22,038	82,727,748	6,924	3,524	255,988	235,008	1,393,932

Results for the numerical solution

Matrix diagrams achieve the highest efficiency in the vector-matrix multiplications...

...and can provide access by columns as required by Gauss–Seidel

Time requirements for the Kanban model

N	$\lvert\mathcal{X}_{reach}\rvert$	number of nonzeros in $\mathbf{R}$	MxDs Gauss–Seidel Iters	MxDs Gauss–Seidel sec/iter	Kronecker Gauss–Seidel Iters	Kronecker Gauss–Seidel sec/iter	Kronecker Jacobi Iters	Kronecker Jacobi sec/iter	Explicit Gauss–Seidel Iters	Explicit Gauss–Seidel sec/iter
2	4,600	28,120	40	0.11	55	0.17	134	0.09	55	0.02
3	58,400	446,400	67	1.46	97	2.56	240	1.34	97	0.34
4	454,475	3,979,850	99	12.33	149	23.69	370	11.99	149	3.04
5	2,546,432	24,460,016	139	73.09	214	147.70	527	74.09	214	18.51
6	11,261,376	115,708,992	185	336.21	289	723.30	713	359.15	—	—
7	41,644,800	450,455,040	238	1,289.91	374	2,922.80	—	—	—	—

Invariant analysis of Petri nets

P-semiflows: Definitions and meaning. Farkas' algorithm. Zero-suppressed integer-range MDDs. A fully symbolic algorithm for p-semiflow computation.

P-semiflows and their generator set

Let $\boxed{\mathbf{D}=\mathbf{D}^{+}-\mathbf{D}^{-}}$ be the flow matrix

A p-semiflow is a non-zero solution $\mathbf{w} \in \mathbb{N}^n$ to the set of linear flow equations $\boxed{\mathbf{w} \cdot \mathbf{D} = \mathbf{0}}$

P-semiflow $\mathbf{w}$ defines the invariant constraint $\boxed{\sum_{p \in \mathcal{P}} \mathbf{w}_p \cdot \boldsymbol{\mu}_p = C}$ on any reachable marking $\boldsymbol{\mu}$

The initial marking $\boldsymbol{\mu}^{init}$ determines the constant $C = \sum_{p \in \mathcal{P}} \mathbf{w}_p \cdot \boldsymbol{\mu}_p^{init}$

P-semiflows provide necessary, not sufficient, conditions on reachability

A linear combination of p-semiflows is a p-semiflow $\Rightarrow$ either none or infinite number of p-semiflows

Support of p-semiflow $\mathbf{w}$: the set of places with positive weight $\boxed{Supp(\mathbf{w}) = \{p \in \mathcal{P} : \mathbf{w}_p > 0\}}$

A p-semiflow $\mathbf{w}$ is minimal if it is scaled back (GCD of its entries is 1) and has minimal support

We seek a (minimal) generator set of minimal p-semiflows $\mathcal{W} = \{\mathbf{w}^{(1)}, ..., \mathbf{w}^{(r)}\}$

Any p-semiflow $\mathbf{w}$ can be derived from $\mathcal{W}$ through non-negative integer linear combinations

The generator set $\mathcal{W}$ is unique, but its size can be exponential in the number of places n

Farkas' algorithm to compute p-semiflows (explicit version)

We manipulate a matrix $[\mathbf{T}|\mathbf{P}]$ stored as a set $\mathcal{A}$ of integer row vectors of length $m+n$

Initially, $[\mathbf{T}|\mathbf{P}] = [\mathbf{D}\,|\,\mathbf{I}] \in \mathbb{Z}^{n \times m} \times \mathbb{N}^{n \times n}$ $\quad$ $\mathbf{D}$ is the flow matrix, $\mathbf{I}$ is the $n \times n$ identity matrix

We iteratively manipulate the set of rows, forcing zero entries in the first j columns, for $j = 1, ..., m$

We substitute rows with positive or negative entry j with linear combinations having a zero entry j

At the end, any row $[\mathbf{t}\,|\,\mathbf{p}]$ in $\mathcal{A}$ is such that $\mathbf{t} \in \mathbb{Z}^m$ is all zero and $\mathbf{p} \in \mathbb{N}^n$ is a p-semiflow

set of array$[m+n]$ of int $ExpPSemiflows$(int n, int m, set of array$[m+n]$ of int $\mathcal{A}$) is

for $j = 1$ to m do
 $\mathcal{A}_N \leftarrow \emptyset$; *set of rows with negative entry j*
 $\mathcal{A}_P \leftarrow \emptyset$; *set of rows with positive entry j*
 foreach $\mathbf{a} \in \mathcal{A}$ do *partition the rows of $\mathcal{A}$ according to entry j*
  if $\mathbf{a}[j] < 0$ then $\mathcal{A}_N \leftarrow \mathcal{A}_N \cup \{\mathbf{a}\}$;
  if $\mathbf{a}[j] > 0$ then $\mathcal{A}_P \leftarrow \mathcal{A}_P \cup \{\mathbf{a}\}$;
 $\mathcal{A} \leftarrow \mathcal{A} \setminus (\mathcal{A}_N \cup \mathcal{A}_P)$; *remove from $\mathcal{A}$ rows with nonzero entry j*
 foreach $\mathbf{a}_N \in \mathcal{A}_N$ and $\mathbf{a}_P \in \mathcal{A}_P$ do *$|\mathcal{A}_N| \cdot |\mathcal{A}_P|$ linear combinations*
  $v \leftarrow MinimumCommonMultiple(-\mathbf{a}_N[j], \mathbf{a}_P[j])$;
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{(-v/\mathbf{a}_N[j]) \cdot \mathbf{a}_N + (v/\mathbf{a}_P[j]) \cdot \mathbf{a}_P\}$; *entry j of new row is 0*
return $\mathcal{A}$;

Compute a generator set (explicit version)

The output $\mathcal{A}$ of Farkas' algorithm may contain **unscaled** and **non-minimal-support** p-semiflows

To scale back the set $\mathcal{A}$ of p-semiflows:

- for each row $\mathbf{a} \in \mathcal{A}$, divide $\mathbf{a}$ by the GCD of all entries in $\mathbf{a}$:
  $\boxed{\mathcal{A} \leftarrow (\mathcal{A} \setminus \{\mathbf{a}\}) \cup \{\mathbf{a}/gcd(\mathbf{a}[m+1], ..., \mathbf{a}[m+n])\}}$
- requires examining each support in $\mathcal{A}$ $\quad$ $\boxed{\text{time complexity } O(|\mathcal{A}| \cdot n)}$

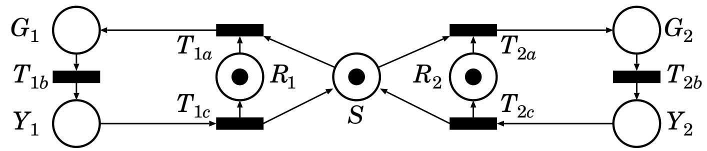
To eliminate from $\mathcal{A}$ the non-minimal-support p-semiflows:

- for each pair of distinct rows $\mathbf{a}$ and $\mathbf{b}$ in $\mathcal{A}$, delete $\mathbf{b}$ if its support is a superset of that of $\mathbf{a}$:
  $\boxed{\text{if } Supp(\mathbf{a}) \subset Supp(\mathbf{b}) \text{ then } \mathcal{A} \leftarrow \mathcal{A} \setminus \{\mathbf{b}\}}$
- requires $|\mathcal{A}| \cdot (|\mathcal{A}| - 1)/2$ support comparisons $\quad$ $\boxed{\text{time complexity } O(|\mathcal{A}|^2 \cdot n)}$

Alternatively, we can avoid adding redundant rows to $\mathcal{A}$ during Algorithm $ExpPSemiflows$:

- before iteration j, $\mathcal{A}$ contains the minimal support p-semiflows ignoring transitions $j, ..., m$
- during iteration j, a row is added to $\mathcal{A}$ only if it does not contain the support of a row in $\mathcal{A}$
- worst-case complexity remains $O(|\mathcal{A}|^2 \cdot n)$, but this alternative is quite beneficial in practice

Example of p-semiflows: a traffic light controller



Initial set of rows $\mathcal{A}$ describing matrix $[\mathbf{T}\,|\,\mathbf{P}] = [\mathbf{D}\,|\,\mathbf{I}]$

	T_{1a}	T_{1b}	T_{1c}	T_{2a}	T_{2b}	T_{2c}	G_1	Y_1	R_1	G_2	Y_2	R_2	S
G_1	1	-1					1						
Y_1		1	-1					1					
R_1	-1		1						1				
G_2				1	-1					1			
Y_2					1	-1					1		
R_2				-1		1						1	
S	-1		1	-1		1							1

Final set of rows $\mathcal{A}$

	T_{1a}	T_{1b}	T_{1c}	T_{2a}	T_{2b}	T_{2c}	G_1	Y_1	R_1	G_2	Y_2	R_2	S
$\mathbf{w}^{(1)}$							1	1	1				
$\mathbf{w}^{(2)}$										1	1	1	
$\mathbf{w}^{(3)}$							1	1		1	1		1

Zero-suppressed extensible multi-way decision diagrams

An MDD over variables $x_L \succ x_{L-1} \succ \cdots \succ x_1$ is a directed acyclic edge-labeled multi-graph:

- A nonterminal node p is associated with a variable $p.var = x_k$, with $L \geq k \geq 1$, and has an infinite set of outgoing edges, each indexed by a different $i \in \mathbb{Z}$
- The only terminal nodes are $\mathbf{0}$ and $\mathbf{1}$ $\quad$ $\boxed{\mathbf{0}.var = \mathbf{1}.var = x_0\text{, with } x_k \succ x_0 \text{ for } L \geq k \geq 1}$
- The edge with index $i \in \mathbb{Z}$ from node p points to a node q $\quad$ $\boxed{p[i] = q\text{, with } p.var \succ q.var}$

Using a zero-suppressed semantic, node p with $p.var = x_k$ encodes the set of k-tuples:

$$\mathcal{X}(p) = \begin{cases} \emptyset & \text{if } p = \mathbf{0} \\ \{\epsilon\} & \text{if } p = \mathbf{1} \\ \bigcup_{i:p[i] \neq \mathbf{0} \wedge p[i].var = x_h} \{i\} \cdot \{0^{k-h-1}\} \cdot \mathcal{X}(p[i]) & \text{otherwise} \end{cases}$$

To enforce a finite representation

- we require that $\{i \in \mathbb{Z} : p[i] \neq \mathbf{0}\}$ be finite

To enforce canonicity

- we forbid nodes where all edges point to $\mathbf{0}$
- we forbid duplicate nodes $\quad$ if $p.var = q.var$ and $p[i] = q[i]$ for all $i \in \mathbb{Z}$, then $p = q$
- we force variable-skipping when possible $\quad$ no node p has $p[i] = \mathbf{0}$ for all $i \in \mathbb{Z} \setminus \{0\}$

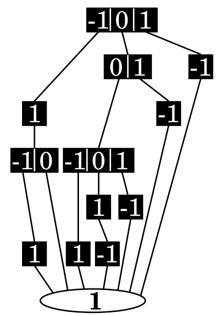
Example: a traffic light controller (continued)

ZMDD encoding the flow matrix for our Petri net:

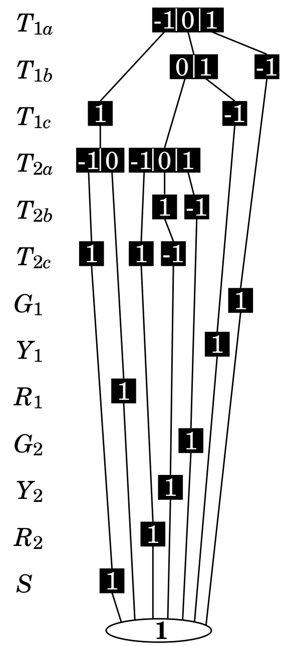
$\mathbf{D} =$

T_{1a}	T_{1b}	T_{1c}	T_{2a}	T_{2b}	T_{2c}
1	-1				
	1	-1			
-1		1			
			1	-1	
				1	-1
			-1		1
-1		1	-1		1

$=$



$[\mathbf{D}|\mathbf{I}] =$



Each row of the matrix is a path in the ZMDD

Terminal node $\mathbf{0}$ and edges pointing to $\mathbf{0}$ are omitted

Observe the skipped nodes due to the zero-suppressed rule

A symbolic approach to compute the generator set

The explicit algorithm to compute the p-semiflows manages sets of row vectors of size $m + n$

Traditional MDDs are ideal to encode and manipulate large sets of same-length tuples

Our ZMDDs excel when most vectors contain mostly zeros, often the case in p-semiflow computation

Our approach is thus:

- Build a ZMDD a over $v_1 \succ \cdots \succ v_m \succ w_1 \succ \cdots \succ w_n$ encoding the set $\mathcal{A}$ of rows in $[\mathbf{D} \mid \mathbf{I}]$
- For $j = 1, ..., m$, symbolically eliminate all rows with $v_j \neq 0$, using linear combinations
  - $\Rightarrow$ After iteration j, all rows have $v_1 = \cdots = v_j = 0$, thus the root of the ZMDD is below v_j
- Apply a symbolic algorithm to eliminate non-minimal support p-semiflows
  - $\Rightarrow$ We propose four variants, differing on when and how rows are eliminated
- At the end, apply a symbolic algorithm to scale back the remaining invariants
  - $\Rightarrow$ We use a symbolic brute force search for the scale-back factors

Pseudo-code for symbolic p-semiflow computation

Given MDD a encoding a set of rows, column $a.var = v_j$ is annulled through the following steps:

- Collect the rows with negative v_j and positive v_j into two new MDDs a_N and a_P
- Leave the rows with $v_j = 0$ in a (being a ZMDD, a becomes $a[0]$, i.e., its height decreases)
- Perform the pairwise linear combination between each $a_P[i_P]$ and $a_N[i_N]$ to annul v_j
- Finally, combine the resulting MDD with a using an ordinary $Union$ operation

```
mdd SymPSemiflows(int m, mdd a) is
for j = 1 to m do
    if a.var ≠ v_j skip;                    nothing to do if a encodes only rows with v_j = 0
    a_N ← Intersection(a, Potential(v_j < 0));             set of rows with negative v_j
    a_P ← Intersection(a, Potential(v_j > 0));             set of rows with positive v_j
    a ← Intersection(a, Potential(v_j = 0));               redefine a for the next iteration
    foreach i_N s.t a_N[i_N] ≠ 0 and i_P s.t. a_P[i_P] ≠ 0 do
        ρ ← MinimumCommonMultiple(−i_N, i_P);
        ρ_N ← ρ/(−i_N);
        ρ_P ← ρ/i_P;
        a ← Union(a, SymLinComb(ρ_N, a_N[i_N], ρ_P, a_P[i_P]));
return a;
```

Symbolic algorithm to perform linear combinations

Given two ZMDDs with $x.var = y.var$, create a ZMDD r with $r.var = x.var = y.var$

For each pair of edges $x[i_x]$ and $y[i_y]$, add a new edge $r[j]$ to r, where $j = \rho_x i_x + \rho_y i_y$

This edge points to the ZMMD node encoding $SymLinComb(\rho_x, x[i_x], \rho_y, y[i_y])$

If r already contains an edge $r[j]$, perform a union instead of creating a new edge

mdd $SymLinComb$(int ρ_x, mdd x, int ρ_y, mdd y) is

- if $x = \mathbf{1}$ and $y = \mathbf{1}$ then return $\mathbf{1}$;
- if $x = \mathbf{0}$ or $y = \mathbf{0}$ then return $\mathbf{0}$;
- if $InCache(C_{SymLinComb}, \rho_x, x, \rho_y, y, r)$ then return r;
- if $x.var \succ y.var$ then — *$y.var$ is skipped*
  - $r \leftarrow NewNode(x.var)$;
  - foreach i_x s.t. $x[i_x] \neq \mathbf{0}$ do $r[\rho_x i_x] \leftarrow SymLinComb(\rho_x, x[i_x], \rho_y, y)$;
- else if $y.var \succ x.var$ then — *$x.var$ is skipped*
  - $r \leftarrow NewNode(y.var)$;
  - foreach i_y s.t. $y[i_y] \neq \mathbf{0}$ do $r[\rho_y i_y] \leftarrow SymLinComb(\rho_x, x, \rho_y, y[i_y])$;
- else — *$y.var = x.var$*
  - $r \leftarrow NewNode(x.var)$;
  - foreach i_x s.t. $x[i_x] \neq \mathbf{0}$ and i_y s.t. $y[i_y] \neq \mathbf{0}$ do
    - $j \leftarrow \rho_x i_x + \rho_y i_y$;
    - $r[j] \leftarrow Union(r[j], SymLinComb(\rho_x, x[i_x], \rho_y, y[i_y]))$
- $r \leftarrow UniqueTableInsert(r)$;
- $CacheAdd(C_{SymLinComb}, \rho_x, x, \rho_y, y, r)$;
- return r;

Removing non-minimal support p-semiflows

The support of a new row is never a subset of the support of a row already in $\mathcal{A}$

new rows never eliminate old rows

Non-minimal-support p-semiflows can be eliminated periodically or at the end

Periodic "internal" elimination, or at the end

- Use a single ZMDD a
- $MinSuppInt$ eliminates p-semiflows that are a linear combination of multiple p-semiflows in a
- The result of $SymLinComb$ is immediately unioned with a
- $MinSuppInt$ can be applied any time to the intermediate result of the p-semiflow computation

Periodic "external" elimination

- ZMDD a encodes a set of minimal-support p-semiflows
- A second ZMDD b stores a single or multiple linear combinations, computed by $SymLinComb$
- $ElimNMSupp$ removes from b p-semiflows with non-minimal-support w.r.t. b
- $MinSuppExt$ removes from b p-semiflows with non-minimal-support w.r.t. a or a and b
- Only then, ZMDDs a and b can be safely unioned

Helper functions to remove non-minimal support p-semiflows

$Prune$ given a ZMDD a, return the $Union$ of all nodes p such that

- p is either a or a descendant of a, and
- either $p.var = w_1$ or $w_1 \succ p.var$ and p is pointed to by an edge from a node q s.t. $q.var \succ w_1$

$MkBool$ given a ZMDD a, return the ZBDD encoding the set of boolean vectors

$$\{\mathbf{b} \in \mathbb{B}^n : \exists \mathbf{x} \in \mathcal{X}(Prune(a)), \forall i, 1 \le i \le n, \mathbf{b}[i] = 1 \Leftrightarrow \mathbf{x}[i] > 0\}$$

$Filter$ given a ZMDD a and a ZBDD b, return the ZMDD encoding

$$\{\mathbf{x} \in \mathcal{X}(Prune(a)) : \exists \mathbf{b} \in \mathcal{X}(b), \forall i, 1 \le i \le n, \mathbf{b}[i] = 1 \Leftrightarrow \mathbf{x}[i] > 0\}$$

An unusual element-wise symbolic operator

$EWOr$ given two ZBDDs p and q, return a ZBDD r of the element-wise-or of all pairs of tuples

$$\mathcal{X}(r) = \{\mathbf{i} \vee \mathbf{j} : \mathbf{i} \in \mathcal{X}(p), \mathbf{j} \in \mathcal{X}(q)\}$$

- quite unlike the much more familiar (non-element-wise) union of sets encoded by two BDDs
- nevertheless, efficient and elegant symbolic implementation

bdd $EWOr$(bdd p, bdd q) is

- if $p = \mathbf{0}$ or $q = \mathbf{0}$ then return $\mathbf{0}$;
- if $p = q$ then return p;
- if $InCache(C_{EWOr}, p, q, r)$ then return r;
- $r \leftarrow NewNode(p.var)$;
- if $p.var \succ q.var$ then $r[0] \leftarrow EWOr(p[0], q)$; $r[1] \leftarrow EWOr(p[1], q)$; — *$q.var$ is skipped*
- if $q.var \succ p.var$ then $r[0] \leftarrow EWOr(p, q[0])$; $r[1] \leftarrow EWOr(p, q[1])$; — *$p.var$ is skipped*
- else — *$p.var = q.var$*
  - $r[0] \leftarrow EWOr(p[0], q[0])$;
  - $r_{01} \leftarrow EWOr(p[0], q[1])$; $r_{10} \leftarrow EWOr(p[1], q[0])$; $r_{11} \leftarrow EWOr(p[1], q[1])$;
  - $r[1] \leftarrow Union(r_{01}, Union(r_{10}, r_{11}))$;
- $r \leftarrow UniqueTableInsert(r)$;
- $CacheAdd(C_{EWOr}, p, q, r)$;
- return r;

The four variants to eliminate non-minimal support rows

mdd $SymPSemiflows$(int m, mdd a) is — *common portion to all variants*

for $j = 1$ to m do
  if $a.var \neq v_j$ skip;
  $a_N \leftarrow Intersection(a, Potential(v_j < 0))$;
  $a_P \leftarrow Intersection(a, Potential(v_j > 0))$;
  $a \leftarrow Intersection(a, Potential(v_j = 0))$;
  $newRows \leftarrow \mathbf{0}$; — *only for V2*
  foreach i_N s.t $a_N[i_N] \neq \mathbf{0}$ and i_P s.t. $a_P[i_P] \neq \mathbf{0}$ do
    $\rho \leftarrow MinimumCommonMultiple(-i_N, i_P)$;
    $\rho_N \leftarrow \rho/(-i_N)$; $\rho_P \leftarrow \rho/i_P$;

V1: Minimize after each linear combination (using external comparisons)

    $linComb = SymLinComb(\rho_N, a_N[i_N], \rho_P, a_P[i_P])$;
    $a \leftarrow Union(a, MinSuppExt(linComb, a))$;
return a;

V2: Minimize after annulling column (using external comparisons)

    $linComb \leftarrow SymLinComb(\rho_N, a_N[i_N], \rho_P, a_P[i_P])$;
    $newRows \leftarrow Union(newRows, linComb)$;
  $a \leftarrow Union(a, MinSuppExt(newRows, a))$;
return a;

V3: Minimize after annulling column (using internal comparisons)

    $a \leftarrow Union(a, SymLinComb(\rho_N, a_N[i_N], \rho_P, a_P[i_P]))$;
  $a \leftarrow MinSuppInt(a)$;
return a;

V4: Minimize only at the end (using internal comparisons)

    $a \leftarrow Union(a, SymLinComb(\rho_N, a_N[i_N], \rho_P, a_P[i_P]))$;
return $MinSuppInt(a)$;

Symbolic scaling of a set of p-semiflows

$SymScalePsemiflows$ repeatedly scales a by the primes in $\{2, ..., \lfloor\sqrt{\text{largest value in } a}\rfloor\}$
The MDDs returned by $ScaleByNumber$ are unioned into the new scaled-back MDD

mdd $SymScalePsemiflows$(mdd a) is
$\gamma \leftarrow \max_{i\in\mathbb{N}}\{p[i] \neq \mathbf{0} : p$ is a node in the MDD $a\}$;
foreach $\mu \in \{2, ..., \lfloor\sqrt{\gamma}\rfloor : \mu$ is prime$\}$ do
  repeat
    $\langle s, u\rangle \leftarrow ScaleByNumber(a, \mu)$; — *$s$ encodes scaled paths, u encodes unscaled paths*
    $a \leftarrow Union(s, u)$; — *update a by combining scaled and unscaled paths*
  until $s = \mathbf{0}$;
return a;

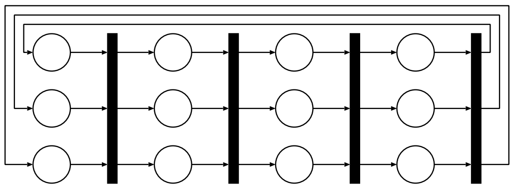
$ScaleByNumber$ takes an MDD a and an integer μ and returns two MDDs:
— s encodes all scaled-back p-semiflows in a which could be scaled by μ
— u encodes those that could not be scaled

$\langle$mdd, mdd$\rangle$ $ScaleByNumber$(mdd a, int μ) is
if $a = \mathbf{0}$ or $a = \mathbf{1}$ then return $\langle a, \mathbf{0}\rangle$;
if $InCache(C_{ScaleByNumber}, a, \mu, \langle s, u\rangle)$ then return $\langle s, u\rangle$;
$s \leftarrow NewNode(a.var)$; — *$s$ will encode paths that were scaled*
$u \leftarrow NewNode(a.var)$; — *$u$ will encode paths that could not be scaled*
foreach i s.t $a[i] \neq \mathbf{0}$ do
  if μ divides i then $\langle s[i/\mu], u[i]\rangle \leftarrow ScaleByNumber(a[i], \mu)$; else $u[i] \leftarrow a[i]$;
$s \leftarrow UniqueTableInsert(s)$;
$u \leftarrow UniqueTableInsert(u)$;
$CacheAdd(C_{ScaleByNumber}, a, \mu, \langle s, u\rangle)$;
return $\langle s, u\rangle$;

Experimental results for p-semiflow computation

The four variants are implemented in SMART and compared with the explicit algorithm in GreatSPN

Experiments run on a Pentium4 3.0GHz PC with 1.0GB of RAM running CentoOS Linux 2.6.9



classic: a classic net with m transitions and m stages of u places each (u^m minimal p-semiflows)

classicX: a modified version of the previous model, using multiple cardinality arcs (still u^m p-semiflows)

trains: a circular railway system with u trains and s rail trunks

slot: a local area network protocol with u nodes in the network

robin: a round robin solution to the mutual exclusion among u processes

aloha: the ALOHA networking protocol on u nodes

mmarch: a multi-threaded architecture with $u \times u$ processing nodes

phil: the classic dining philosophers problem, with u philosophers

power: power distribution system with u generators (one p-semiflow)

Models requiring minimal-support elimination

model	trans	p-semiflows	nodes	edges		mem	PS	MS	SB	time
trains10	100	37	642	678	V1	6	5.02%	94.95%	0.03%	4.70
					V2	6	4.99%	94.98%	0.03%	4.81
					V3	7	4.76%	95.17%	0.07%	4.05
					V4	om				
					GS	0.06	-	-	-	0.03
trains15	255	99	3,533	3,620	V1	19	4.86%	95.13%	0.01%	210.60
					V2	19	4.80%	95.17%	0.03%	210.05
					V3	21	4.98%	95.01%	0.01%	216.30
					V4	om				
					GS	0.242	-	-	-	0.33
slot20	160	42	1,597	1,798	V1	58	0.20%	99.79%	0.01%	57.44
					V2	58	0.20%	99.79%	0.01%	57.46
					V3	om				
					V4	5	98.46%	0.01%	1.53%	0.29
					GS	3	-	-	-	0.08
slot2000	16,000	4,002	31,997	35,998	V1	om				
					V2	om				
					V3	om				
					V4	242	99.64%	0.01%	0.36%	126.12
					GS	876	-	-	-	189.02

Models requiring minimal-support elimination (cont.)

model	trans	p-semiflows	nodes	edges		mem	PS	MS	SB	time
robin4	24	30	78	96	V1	0.198	15.11%	83.45%	1.44%	0.012
					V2	0.198	15.06%	83.51%	1.42%	0.012
					V3	0.187	10.76%	88.14%	1.10%	0.015
					V4	26	99.9%	0%	0%	11.06
					GS	3	-	-	-	0.01
robin90	540	1.24×10^{27}	1,798	2,160	V1	57	1.02%	99.97%	0.01%	64.89
					V2	57	0.36%	99.62%	0.01%	64.81
					V3	om				
					V4	om				
					GS	ot				
aloha15	60	32,771	78	96	V1	0.325	30.17%	68.99%	0.83%	0.02
					V2	0.326	30.31%	68.88%	0.81%	0.02
					V3	0.362	17.25%	82.26%	0.49%	0.03
					V4	om				
					GS	4	-	-	-	33.20
aloha100	400	1.27×10^{30}	503	606	V1	12	43.42%	56.43%	0.14%	1.00
					V2	12	43.91%	55.95%	0.14%	1.02
					V3	14	6.21%	93.76%	0.03%	4.96
					V4	om				
					GS	ot				

Models not requiring minimal-support elimination

model	trans	p-semiflows	nodes	edges		mem	PS	MS	SB	time
classic10	10	1×10^{10}	100	190	V1	0.055	16.19%	75.17%	8.64%	0.0027
					V2	0.055	15.99%	75.69%	8.31%	0.0028
					V3	0.058	8.51%	87.15%	4.33%	0.0054
					V4	0.031	80.66%	0.81%	18.53%	0.0014
					GS	ot				
classic250	250	3.05×10^{599}	62,500	124,750	V1	613	0.58%	99.00%	0.37%	54.20
					V2	613	0.58%	99.05%	0.37%	53.59
					V3	om				
					V4	8	84.97%	0.01%	15.03%	0.92
					GS	ot				
mmarch10	1,400	404	1,200	1,603	V1	40	0.82%	99.12%	0.06%	4.76
					V2	40	0.83%	99.11%	0.06%	4.79
					V3	om				
					V4	2	95.85%	0.05%	4.93%	0.0056
					GS	3	-	-	-	0.01
mmarch20	5,600	1,604	4,800	6,403	V1	198	3.84%	96.15%	0.01%	122.15
					V2	198	3.91%	96.08%	0.01%	121.99
					V3	om				
					V4	6	96.00%	0%	3.99%	0.32
					GS	110	-	-	-	4.29

Models not requiring minimal-support elimination (cont.)

model	trans	p-semiflows	nodes	edges		mem	PS	MS	SB	time
phil30	120	90	239	328	V1	11	2.35%	97.59%	0.06%	1.04
					V2	11	2.36%	97.59%	0.06%	1.04
					V3	11	0.43%	99.53%	0.04%	1.61
					V4	0.346	94.45%	0.17%	5.38%	0.011
					GS	0.1	-	-	-	0.01
phil100	400	300	799	1,098	V1	50	0.19%	99.79%	0.01%	30.90
					V2	50	0.19%	99.80%	0.01%	30.85
					V3	50	0.13%	99.86%	0.01%	47.64
					V4	2	96.82%	0.04%	3.13%	0.077
					GS	1	-	-	-	0.04
power50	2,600	1	51	51	V1	2	51.25%	48.68%	0.06%	0.21
					V2	2	51.17%	48.76%	0.06%	0.21
					V3	2	24.03%	75.94%	0.03%	0.45
					V4	2	99.84%	0.02%	0.13%	0.11
					GS	0.6	-	-	-	0.08
power100	10,200	1	101	101	V1	14	51.39%	48.58%	0.01%	2.05
					V2	14	51.37%	48.62%	0.01%	2.04
					V3	14	23.51%	76.49%	0.01%	4.50
					V4	14	99.98%	0.00%	0.01%	1.05
					GS	4	-	-	-	0.64
classicX8	8	1.67×10^{6}	5,193	9,402	V1	14	0.31%	99.87%	0.01%	273.42
					V2	2	68.60%	17.26%	13.94%	0.27
					V3	2	70.50%	16.33%	13.17%	0.29
					V4	4	86.47%	0.01%	13.53%	0.28
					GS	ot				
classicX12	12	8.92×10^{12}	61,584	114,648	V1	om				
					V2	18	84.43%	9.54%	6.02%	29.73
					V3	19	82.78%	11.05%	6.16%	29.07
					V4	18	93.71%	0%	6.29%	28.31
					GS	ot				

Conclusions from the numerical results

For models whose runtime is greater than 1 sec, less than 1% of runtime is spent scaling back
Except for *classicX*, which requires many $ScaleByNumber$ calls due to the large row entries
Even on *classicX*, scaling back requires less than 14% of runtime

Best: V1,V2,V3 (*trains*), V4 (*slot*,*classic*,*mmarch*,*phil*,*power*) V1,V2 (*robin*,*aloha*) V2,V3,V4 (*classicX*)
At least one of either V2 or V4 was the most efficient (or very close) for each model
V2 works best for models that add many non-minimal support p-semiflows at each step
V4 works best for models where few non-minimal support p-semiflows are generated
We could start with V4 and switch to V2 if many non-minimal support p-semiflows are being generated
Alternatively, we could run V2 and V4 on two independent workstations

GreatSPN tends to be more efficient for models with relatively few p-semiflows
For most models, at least one of our variants outperforms GreatSPN for large enough instances

Our new symbolic method offers vast time and space improvements
— the most dramatic example is *classic250*: 3.05×10^{599} p-semiflows in 1 sec using 8MBytes
— many Petri nets with unit arc cardinalities require a single call to $SymLinComb$ per column

Two types of models have larger time and memory requirements with our symbolic method
— models with a dense flow matrix, they do not benefit as much from the ZMDD properties
— models with arc cardinalities greater than one, as revealed by comparing *classic* and *classicX*
(still, the symbolic method can generate the 8.92×10^{12} p-semiflows of *classicX12* in 30 sec)

Relevant references from my work

[1] P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. *INFORMS J. Comp.*, 12(3):203–222, 2000.

[2] M.-Y. Chung, G. Ciardo, and A. J. Yu. A fine-grained fullness-guided chaining heuristic for symbolic reachability analysis. In S. Graf and W. Zhang, editors, *Proc. 4th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, LNCS 4218, pages 51–66, Beijing, China, Oct. 2006. Springer-Verlag.

[3] G. Ciardo. Data representation and efficient solution: a decision diagram approach. In M. Bernardo and J. Hillston, editors, *Formal Methods for Performance Evaluation*, LNCS 4486, pages 371–394, Bertinoro, Italy, May 2007. Springer-Verlag.

[4] G. Ciardo, M. Forno, P. L. Grieco, and A. S. Miner. Comparing implicit representations of large CTMCs. In W. J. Stewart and A. N. Langville, editors, *Numerical Solution of Markov Chains*, pages 323–327, Urbana, IL, USA, Sept. 2003.

[5] G. Ciardo, G. Lüttgen, and A. S. Miner. Exploiting interleaving semantics in symbolic state-space generation. *Formal Methods in System Design*, 31:63–100, 2007.

[6] G. Ciardo, G. Lüttgen, and R. Siminiceanu. Saturation: An efficient iteration strategy for symbolic state space generation. In T. Margaria and W. Yi, editors, *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS 2031, pages 328–342, Genova, Italy, Apr. 2001. Springer-Verlag.

[7] G. Ciardo, R. Marmorstein, and R. Siminiceanu. Saturation unbound. In H. Garavel and J. Hatcliff, editors, *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS 2619, pages 379–393, Warsaw, Poland, Apr. 2003. Springer-Verlag.

[8] G. Ciardo, R. Marmorstein, and R. Siminiceanu. The saturation algorithm for symbolic state space exploration. *Software Tools for Technology Transfer*, 8(1):4–25, Feb. 2006.

[9] G. Ciardo, G. Mecham, E. Paviot-Adet, and M. Wan. P-semiflow computation with decision diagrams. In G. Franceschinis and K. Wolf, editors, *Proc. 30th International Conference on Application and Theory of Petri nets and Other Models of Concurrency (ICATPN)*, LNCS 5606, pages 143–162, Paris, France, June 2007. Springer-Verlag.

[10] G. Ciardo and A. S. Miner. Implicit data structures for logic and stochastic systems analysis. *ACM SIGMETRICS Perf. Eval. Rev.*, 32(4):4–9, 2005.

[11] G. Ciardo and R. Siminiceanu. Using edge-valued decision diagrams for symbolic generation of shortest paths. In M. D. Aagaard and J. W. O'Leary, editors, *Proc. Fourth International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, LNCS 2517, pages 256–273, Portland, OR, USA, Nov. 2002. Springer-Verlag.

[12] G. Ciardo and R. Siminiceanu. Structural symbolic CTL model checking of asynchronous systems. In W. Hunt, Jr. and F. Somenzi, editors, *Computer Aided Verification (CAV'03)*, LNCS 2725, pages 40–53, Boulder, CO, USA, July 2003. Springer-Verlag.

[13] G. Ciardo and A. J. Yu. Saturation-based symbolic reachability analysis using conjunctive and disjunctive partitioning. In D. Borrione and W. Paul, editors, *Proc. CHARME*, LNCS 3725, pages 146–161, Saarbrücken, Germany, Oct. 2005. Springer-Verlag.

[14] M. Wan and G. Ciardo. Symbolic reachability analysis of integer timed Petri nets. In M. Nielsen et al., editors, *Proc. 35th Int. Conf. Current Trends in Theory and Practice of Computer Science (SOFSEM)*, LNCS 5404, pages 595–608, Špindlerův Mlýn, Czech Republic, Feb. 2009. Springer-Verlag.

[15] M. Wan and G. Ciardo. Symbolic state-space generation of asynchronous systems using extensible decision diagrams. In M. Nielsen et al., editors, *Proc. 35th Int. Conf. Current Trends in Theory and Practice of Computer Science (SOFSEM)*, LNCS 5404, pages 582–594, Špindlerův Mlýn, Czech Republic, Feb. 2009. Springer-Verlag.