Reachability Set Generation for Petri Nets: Can Brute Force Be Smart?

Gianfranco Ciardo

Department of Computer Science and Engineering
University of California at Riverside
Riverside, CA 92521, USA
ciardo@cs.ucr.edu

Andrew S. Miner (William and Mary, VA, USA $\Rightarrow$ Iowa State University, IA, USA)
Radu Siminiceanu (William and Mary, VA, USA $\Rightarrow$ National Institute for Aerospace, VA, USA)
Gerald Luettgen (ICASE, VA, USA $\Rightarrow$ Sheffield University, UK $\Rightarrow$ York University, UK)
Ming Ying Chung, Ning He, Jinqing Yu

Work supported in part by NSF grants CCR-0219745 and ACI-0203971 and NASA grant NAG-1-02095

Outline

- Discrete-state models and their state space
- Self-modifying Petri nets
- Explicit vs. symbolic state-space generation
- Two heuristics to improve symbolic state-space generation

- MDDs for the state space and Kronecker for the next-state function
- Locality and *saturation*
- Other uses of the saturation algorithm
- Ongoing work

Structured discrete-state models

A *structured discrete state model* is specified by

- a *potential state space* $\widehat{\mathcal{S}} = \mathcal{S}_K \times \cdots \times \mathcal{S}_1 = \times_{K \geq k \geq 1} \mathcal{S}_k$
  - the "type" of the (global) state
  - $\mathcal{S}_k$ is the (discrete) *local state space* for submodel k
- a set of *initial states* $\mathcal{S}^{init} \subseteq \widehat{\mathcal{S}}$
  - often there is a single initial state $\mathbf{s}^{init}$
- a set of *events* $\mathcal{E}$ defining a *disjunctively-partitioned next-state function*
  - $\mathcal{N}_\alpha : \widehat{\mathcal{S}} \rightarrow 2^{\widehat{\mathcal{S}}}$ $\quad\quad$ $\mathbf{j} \in \mathcal{N}_\alpha(\mathbf{i})$ iff state $\mathbf{j}$ can be reached by *firing* event α in state $\mathbf{i}$
  - $\mathcal{N} : \widehat{\mathcal{S}} \rightarrow 2^{\widehat{\mathcal{S}}}$ is defined by $\mathcal{N}(\mathbf{i}) = \bigcup_{\alpha \in \mathcal{E}} \mathcal{N}_\alpha(\mathbf{i})$
  - we can extend $\mathcal{N}$ to take sets of states as argument $\mathcal{N}(\mathcal{X}) = \bigcup_{\mathbf{i} \in \mathcal{X}} \mathcal{N}(\mathbf{i})$
  - α is *enabled* in $\mathbf{i}$ iff $\mathcal{N}_\alpha(\mathbf{i}) \neq \emptyset$, otherwise it is *disabled*
  - $\mathbf{i}$ is *absorbing*, or a *trap*, or *dead* iff $\mathcal{N}(\mathbf{i}) = \emptyset$

The state space of a discrete-state model

The *state space* $\mathcal{S}$ of the model is the smallest subset of $\widehat{\mathcal{S}}$ containing $\mathcal{S}^{init}$ and satisfying:

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

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

Self-modifying Petri nets with inhibitor arcs

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

- $\mathcal{P}$ and $\mathcal{T}$ — places and transitions
- $\mathbf{F}^-, \mathbf{F}^+ : \mathcal{P} \times \mathcal{T} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N}$ — marking-dependent input, output arc cardinalities
- $\mathbf{F}^\circ : \mathcal{P} \times \mathcal{T} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N} \cup \{\infty\}$ — 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 $\forall p \in \mathcal{P},\ \mathbf{F}^-_{p,\alpha}(\mathbf{i}) \leq \mathbf{i}_p \ \wedge\ \mathbf{F}^\circ_{p,\alpha}(\mathbf{i}) > \mathbf{i}_p$

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

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

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

Explicit vs. symbolic state-space generation

State-by-state (explicit) generation algorithm

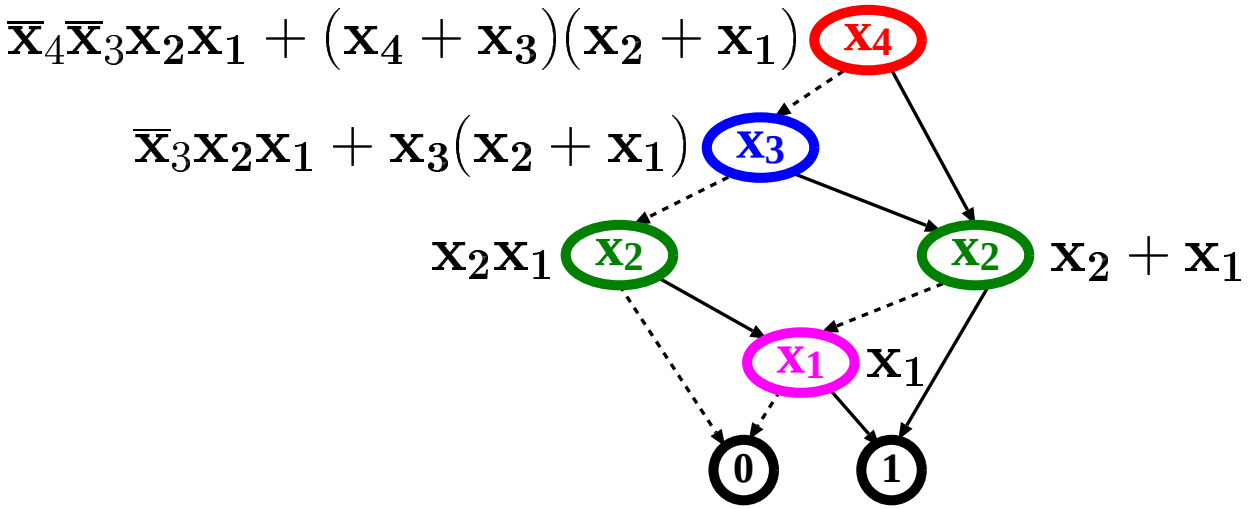
$ExploreExplicit(\mathcal{S}^{init}, \mathcal{N})$ is

1. $\mathcal{S} \leftarrow \emptyset$; — *$\mathcal{S}$ contains the known states already explored*
2. $\mathcal{U} \leftarrow \{\mathcal{S}^{init}\}$; — *$\mathcal{U}$ contains the known states not yet explored*
3. while $\mathcal{U} \neq \emptyset$ do
4. choose a state $\mathbf{i}$ in $\mathcal{U}$ and move it from $\mathcal{U}$ to $\mathcal{S}$;
5. for each $\mathbf{j} \in \mathcal{N}(\mathbf{i})$ do
6. if $\mathbf{j} \notin \mathcal{S} \cup \mathcal{U}$ 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 $\mathcal{S}$;

the most expensive operation is searching for a state (line 6)

BDDs: (Reduced ordered) binary decision diagrams

- There is a single root node r
- Each non-terminal node is labeled with a boolean variable $\mathbf{x}_k \in \{\mathbf{x}_K, ..., \mathbf{x}_1\}$
- Terminal nodes are labeled 0 or 1
- A non-terminal node has two outgoing arcs, labeled 0 (dashed) and 1 (solid)
- An arc from a node labeled $\mathbf{x}_k$ points to a node labeled $\mathbf{x}_l$, $k > l$
- Two nodes labeled $\mathbf{x}_k$ cannot have the same pattern of children *(no duplicates)*
- The two children of a node are different *(no redundant nodes)*



canonical representation of boolean functions $\{0,1\}^K \rightarrow \{0,1\}$

"Graph-based algorithms for boolean function manipulation"

Bryant (IEEE TC, 1986) one of CiteSeer most cited documents!

Using BDDs for state space generation

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

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

$2K$-level BDD encodes the *transition relation* $\mathcal{N} \subseteq \widehat{\mathcal{S}} \times \widehat{\mathcal{S}}$

$(\mathbf{i}, \mathbf{j}) \equiv (\mathbf{i}_K, \mathbf{j}_K, ..., \mathbf{i}_1, \mathbf{j}_1) \in \mathcal{N} \Leftrightarrow$ the system can go from $\mathbf{i}$ to $\mathbf{j}$ in one step

We can also think of it as the *next-state function* $\mathcal{N} : \widehat{\mathcal{S}} \to 2^{\widehat{\mathcal{S}}}$

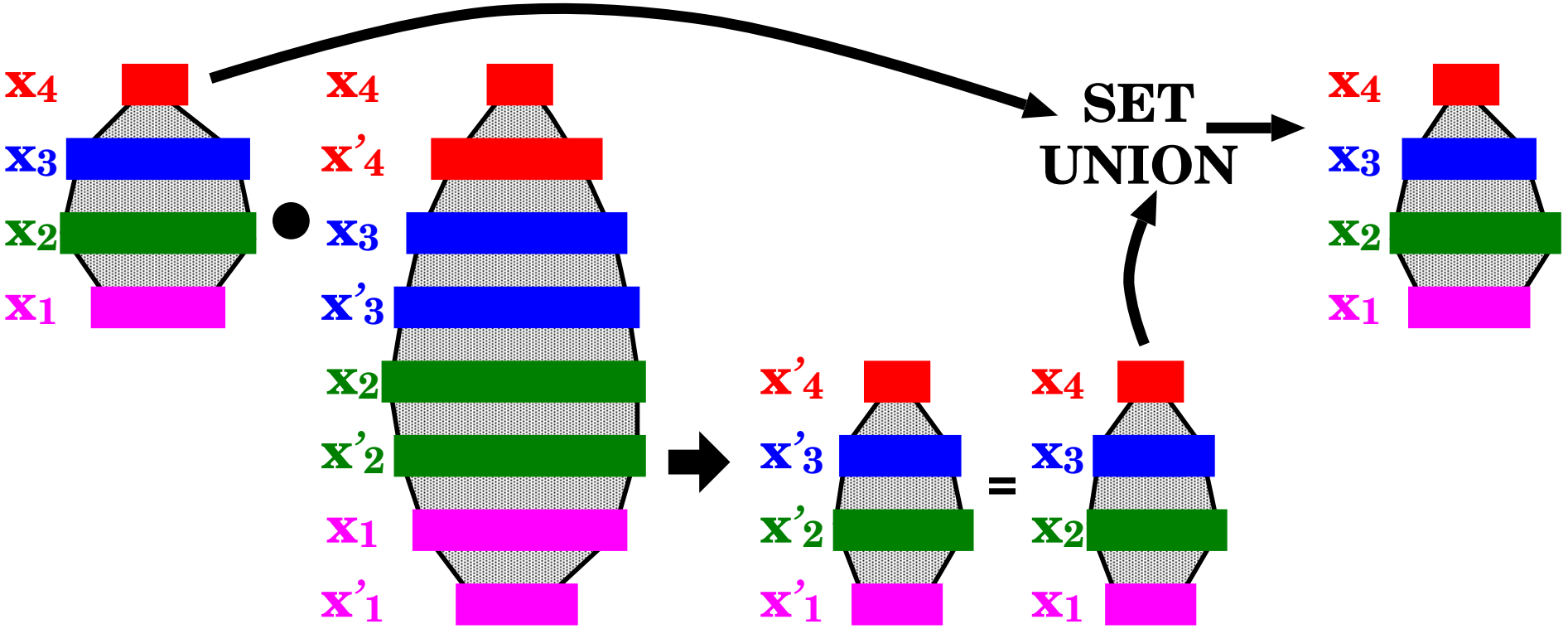
$\mathbf{j} \in \mathcal{N}(\mathbf{i}) \Leftrightarrow$ the system can go from $\mathbf{i}$ to $\mathbf{j}$ in one step

$BfSsGen(\mathcal{S}^{init}, \mathcal{N})$ is

1. $\mathcal{S} \leftarrow \mathcal{S}^{init}$; *known states*
2. $\mathcal{U} \leftarrow \mathcal{S}^{init}$; *unexplored states*
3. while $\mathcal{U} = \emptyset$ do
4. $\quad \mathcal{X} \leftarrow \mathcal{N}(\mathcal{U})$; *potentially new states*
5. $\quad \mathcal{U} \leftarrow \mathcal{X} \setminus \mathcal{S}$; *truly new states*
6. $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{U}$;
7. return $\mathcal{S}$;

BDD encoding of the next-state function

- Given a current set of states $\mathcal{X}$ encoded as a BDD in K variables $(\mathbf{x}_K, \ldots, \mathbf{x}_1)$ and
- the *next-state function* $\mathcal{N}$ encoded as a BDD in $2K$ variables $(\mathbf{x}_K, \mathbf{x}'_K \ldots, \mathbf{x}_1, \mathbf{x}'_1)$
- we obtain the set of states $\mathcal{N}(\mathcal{X})$ reachable from $\mathcal{X}$ in one step



iterations to generate the state space $\mathcal{S}$: max distance d of any state from initial states + 1

peak BDD size usually occurs well before reaching the *final* BDD for $\mathcal{S}$, at step d

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

We can store

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

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

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

The *topological image computation* for a transition α 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

Chaining [Roig95]

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

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

is exactly achieved with the statements

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

If we don't require strict breadth-first order, however, we can use *chaining*

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

Using new states vs. all states in symbolic iterations

$BfSsGen(\mathcal{S}^{init}, \mathcal{N})$

1. $\mathcal{S} \leftarrow \mathcal{S}^{init}$;
2. $\mathcal{U} \leftarrow \mathcal{S}^{init}$;
3. while $\mathcal{U} \neq \emptyset$ do
4. $\quad \mathcal{X} \leftarrow \mathcal{N}(\mathcal{U})$;
5. $\quad \mathcal{U} \leftarrow \mathcal{X} \setminus \mathcal{S}$;
6. $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{U}$;
7. return $\mathcal{S}$;

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

1. $\mathcal{S} \leftarrow \mathcal{S}^{init}$;
2. $\mathcal{U} \leftarrow \mathcal{S}^{init}$;
3. while $\mathcal{U} \neq \emptyset$ do
4. $\quad$ for each $\alpha \in \mathcal{E}$ do
5. $\quad\quad \mathcal{U} \leftarrow \mathcal{U} \cup \mathcal{N}_\alpha(\mathcal{U})$;
6. $\quad \mathcal{U} \leftarrow \mathcal{U} \setminus \mathcal{S}$;
7. $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{U}$;
8. return $\mathcal{S}$;

$AllBfSsGen(\mathcal{S}^{init}, \mathcal{N})$

1. $\mathcal{S} \leftarrow \mathcal{S}^{init}$;
2. $\mathcal{O} \leftarrow \emptyset$;
3. while $\mathcal{O} \neq \mathcal{S}$ do
4. $\quad \mathcal{O} \leftarrow \mathcal{S}$;
5. $\quad \mathcal{S} \leftarrow \mathcal{O} \cup \mathcal{N}(\mathcal{O})$;
6. return $\mathcal{S}$;

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

1. $\mathcal{S} \leftarrow \mathcal{S}^{init}$;
2. $\mathcal{O} \leftarrow \emptyset$;
3. while $\mathcal{O} \neq \mathcal{S}$ do
4. $\quad \mathcal{O} \leftarrow \mathcal{S}$;
5. $\quad$ for each $\alpha \in \mathcal{E}$ do
6. $\quad\quad \mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{N}_\alpha(\mathcal{S})$;
7. return $\mathcal{S}$;

Comparing the four approaches

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

MDDs for $\mathcal{S}$ and Kronecker for $\mathcal{N}$

(Quasi-reduced ordered) multi-way decision diagrams

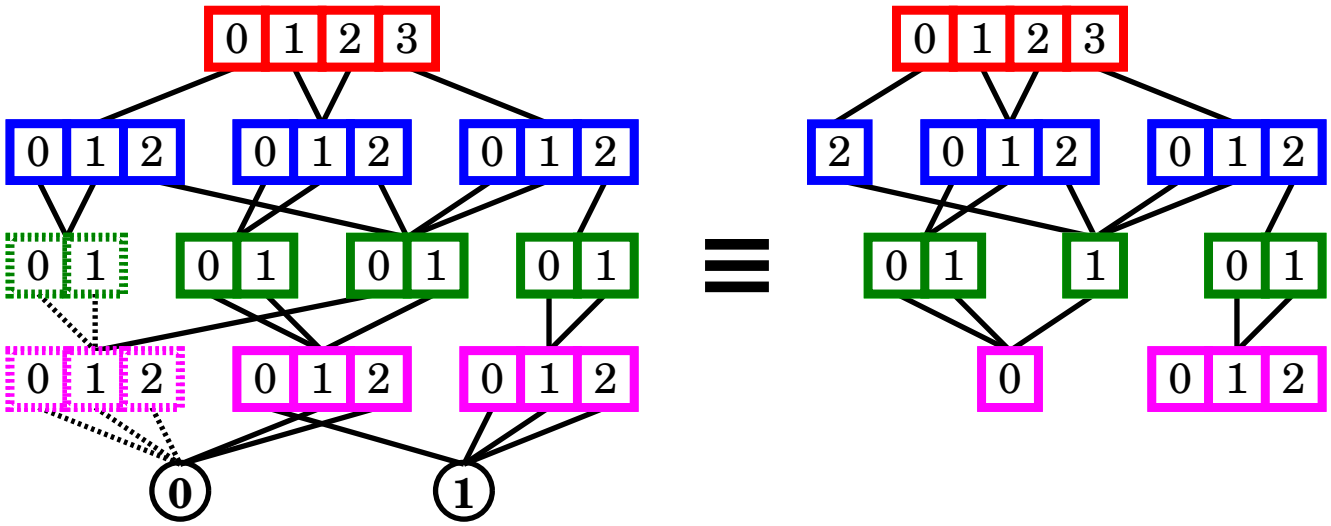
- Nodes are organized into $K+1$ **levels**
  - Level K contains only one **root** node
  - Levels $K-1$ through 1 contain one or more nodes, no **duplicates**
  - Level 0 contains the only two **terminal nodes**, $\mathbf{0}$ and $\mathbf{1}$, corresponding to **false** and **true**
- For $k > 0$, a node at level k has $|\mathcal{S}_k|$ arcs pointing to nodes at level $k-1$

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

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

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

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



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

[Kam 1998] defi ned fully-reduced MDDs as an interface to BDDs

Kronecker-consistent decomposition of a model

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

- $\mathcal{N}$ is disjunctively partitioned according to a set of *events* $\mathcal{E}$ $\boxed{\mathcal{N}(\mathbf{i}) = \bigcup_{\alpha\in\mathcal{E}} \mathcal{N}_\alpha(\mathbf{i})}$
- $\widehat{\mathcal{S}} = \times_{K\geq k\geq 1}\mathcal{S}_k$, a *global* state $\mathbf{i}$ consists of K *local* states $\boxed{\mathbf{i} = (\mathbf{i}_K, \ldots, \mathbf{i}_1)}$
- and, most importantly, we can write $\boxed{\mathcal{N}_\alpha(\mathbf{i}) = \times_{K\geq k\geq 1}\mathcal{N}_{k,\alpha}(\mathbf{i}_k)}$

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

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

We encode the next state function with $K \cdot |\mathcal{E}|$ small matrices $\mathbf{N}_{k,\alpha} \in \{0,1\}^{|\mathcal{S}_k\times\mathcal{S}_k|}$

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

(even with inhibitor, reset, or probabilistic arcs)

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

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

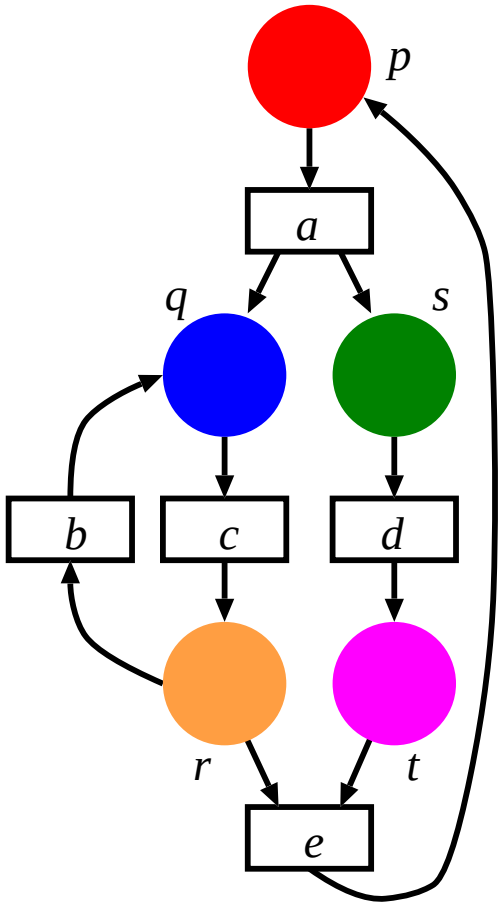
EVENTS →

LEVELS ↓

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

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

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



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

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

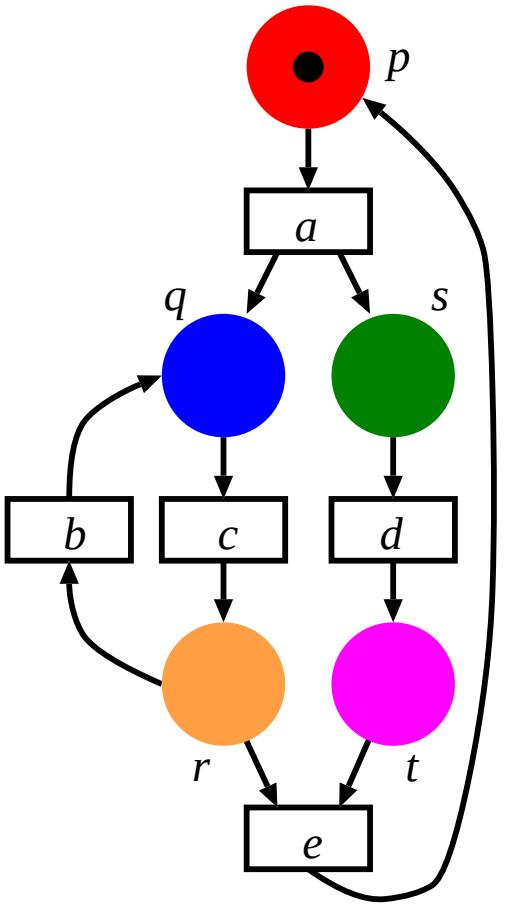
$\mathcal{S}_5 : \{p^1,p^0\}\equiv\{0,1\}$ $\mathcal{S}_4 : \{q^0,q^1\}\equiv\{0,1\}$ $\mathcal{S}_3 : \{r^0,r^1\}\equiv\{0,1\}$ $\mathcal{S}_2 : \{s^0,s^1\}\equiv\{0,1\}$ $\mathcal{S}_1 : \{t^0,t^1\}\equiv\{0,1\}$

EVENTS →

LEVELS ↓

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

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



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

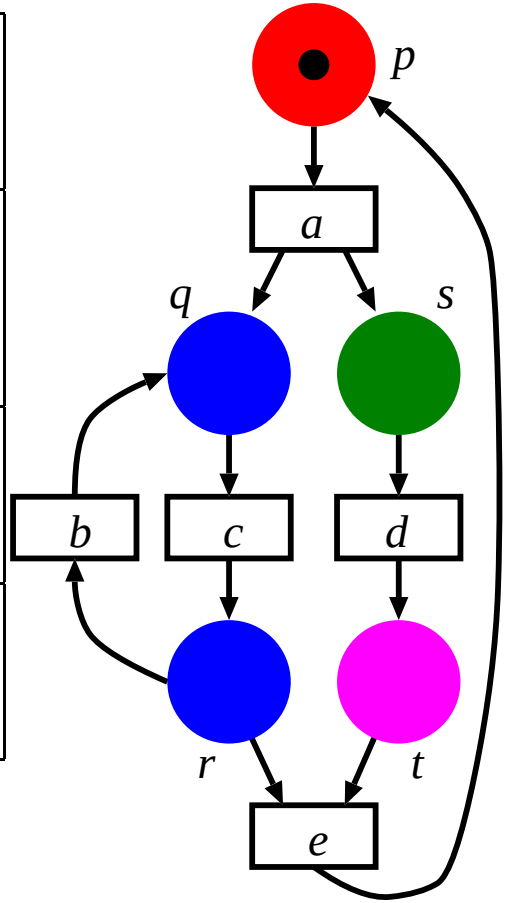
$\mathcal{S}_4 : \{p^1, p^0\} \equiv \{0,1\}$ $\mathcal{S}_3 : \{q^0r^0, q^1r^0, q^0r^1\} \equiv \{0,1,2\}$ $\mathcal{S}_2 : \{s^0, s^1\} \equiv \{0,1\}$ $\mathcal{S}_1 : \{t^0, t^1\} \equiv \{0,1\}$

EVENTS $\longrightarrow$

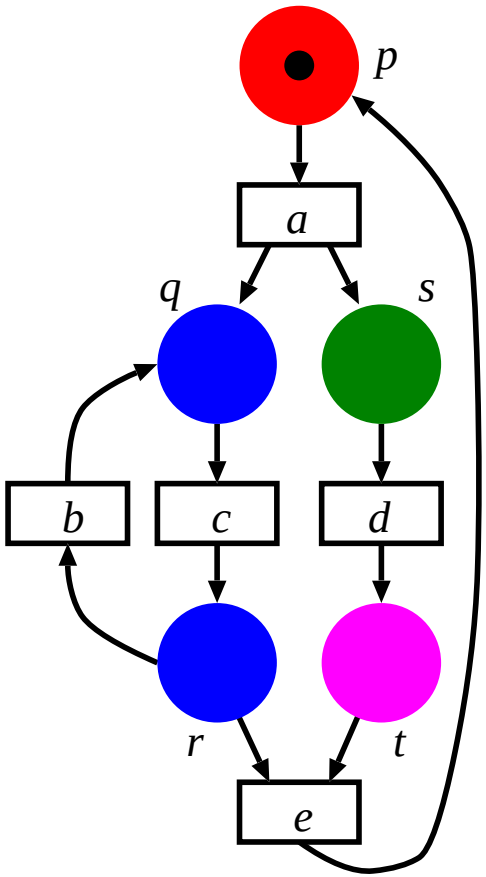
LEVELS $\downarrow$

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

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



The matrix N encoded by the Kronecker descriptor $(K = 4)$

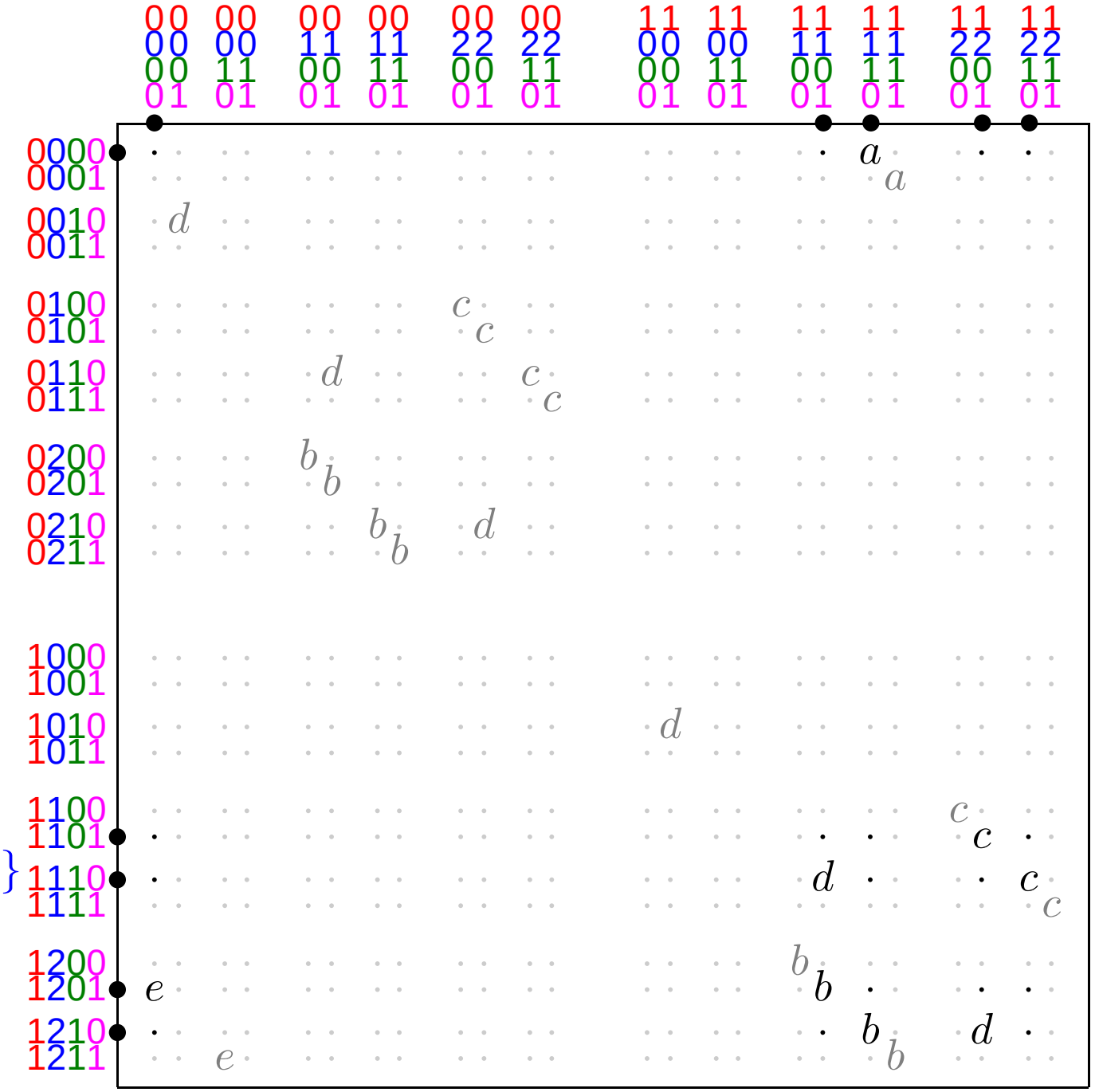


$\{p^1,p^0\} \equiv \{0,1\}$

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

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

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



Locality and Saturation

Locality

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

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

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

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

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

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

Exploiting locality

Locality restricts the range of levels to which $\mathcal{N}_\alpha$ is applied:

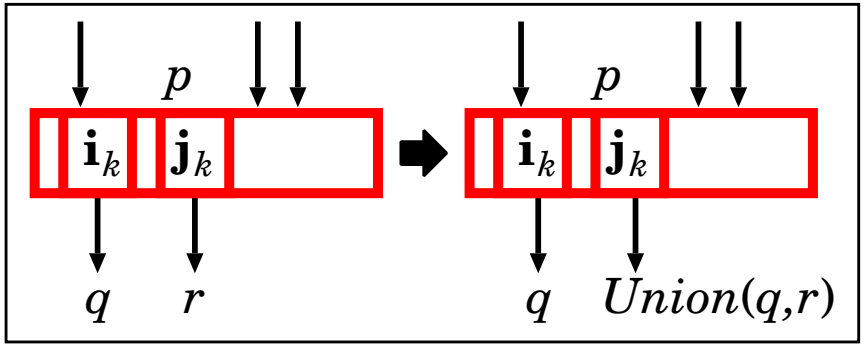
If $\mathbf{i} \in \mathcal{S}$, $\mathbf{j} \in \mathcal{N}_\alpha(\mathbf{i})$, $Top(\alpha) = k \wedge Bot(\alpha) = l$: $\quad \mathbf{j} = (\mathbf{i}_K, ..., \mathbf{i}_{k+1}, \mathbf{j}_k, ..., \mathbf{j}_l, \mathbf{i}_{l-1}, ..., \mathbf{i}_1)$

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

If $\mathbf{i}' = (\mathbf{i}'_K, ..., \mathbf{i}'_{k+1}, \mathbf{i}_k, ..., \mathbf{i}_1) \in \mathcal{S}$: $\quad \mathbf{j}' \in \mathcal{N}_\alpha(\mathbf{i}') \wedge \mathbf{j}' = (\mathbf{i}'_K, ..., \mathbf{i}'_{k+1}, \mathbf{j}_k, ..., \mathbf{j}_l, \mathbf{i}_{l-1}, ..., \mathbf{i}_1)$

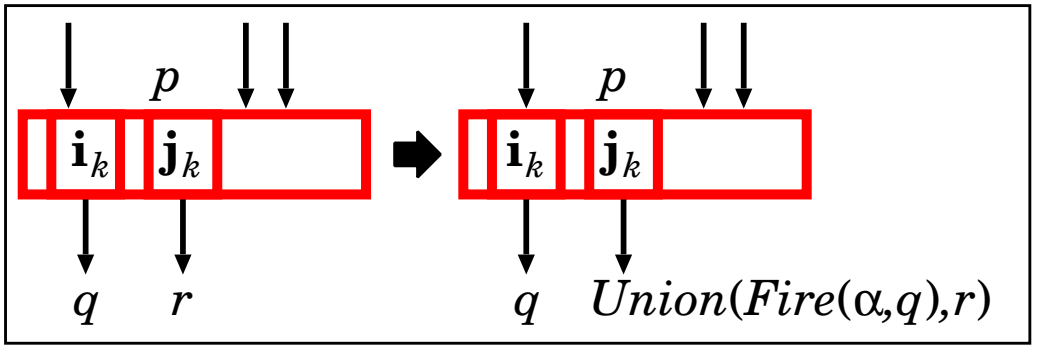
$Top(\alpha) = Bot(\alpha)$

Local event α: $\mathbf{i}_k \xrightarrow{\alpha} \mathbf{j}_k$



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

Synchronizing event α: $(\mathbf{i}_k, ..., \mathbf{i}_l) \xrightarrow{\alpha} (\mathbf{j}_k, ..., \mathbf{j}_l)$



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

[work presented at ICATPN 2000]

Saturation: an iteration strategy based on the model structure

MDD node p at level k is ***saturated*** if the set of states it encodes is a fi xed point w.r.t. any α s.t. $Top(\alpha) \leq k$ (thus, all nodes below p are also saturated)

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

states are **not** discovered in breadth-fi rst order

enormous time and memory savings for

globally-asynchronous locally-synchronous systems

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
- We must consider redundant nodes as well, thus the use of quasi-reduced MDDs
- Once node p at level k is saturated, we never fire any event α s.t. $k = Top(\alpha)$ in 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

enourmous reductions in the peak number of nodes

enourmous reductions in the runtimes

Solution requirements: SMART vs. NuSMV (800MHz P-III)

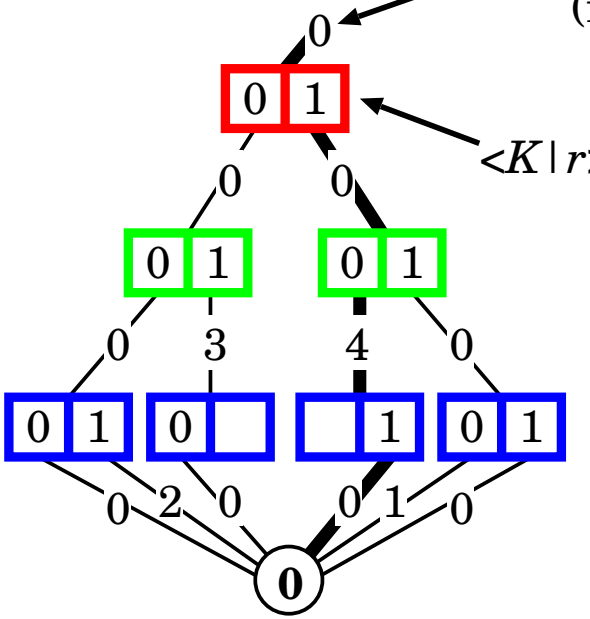
Time and memory to generate the state space of various parametric models

N	$\|\mathcal{S}\|$	Final memory (kB) SMART	Final memory (kB) NuSMV	Peak memory (kB) SMART	Peak memory (kB) NuSMV	Time (sec) SMART	Time (sec) NuSMV
Dining Philosophers: $K = N$							
50	2.23×10^{31}	18	10,800	22	10,819	0.15	5.9
200	2.47×10^{125}	74	27,155	93	72,199	0.68	12,905.7
10,000	4.26×10^{6269}	3,749	—	4,686	—	877.82	—
Slotted Ring Network: $K = N$							
10	8.29×10^{9}	4	5,287	28	10,819	0.13	5.5
15	1.46×10^{15}	10	9,386	80	13,573	0.39	2,039.5
200	8.38×10^{211}	1,729	—	120,316	—	902.11	—
Round Robin Mutual Exclusion: $K = N + 1$							
20	4.72×10^{7}	18	7,300	20	7,306	0.07	0.8
100	2.85×10^{32}	356	16,228	372	26,628	3.81	2,475.3
300	1.37×10^{93}	3,063	—	3,109	—	140.98	—
Flexible Manufacturing System: $K = 19$							
10	4.28×10^{6}	16	1,707	26	11,238	0.05	9.4
20	3.84×10^{9}	55	14,077	101	31,718	0.20	1,747.8
250	3.47×10^{26}	25,507	—	69,087	—	231.17	—

Other uses of the saturation algorithm

Storing partial arithmetic functions with EV$^+$MDDs

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



ρ value associated with the root (minimum value of the function)

<K | r> root node r, at level K = 3

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

If $(\rho, \langle K|r \rangle)$ encodes f, then $\rho = \min\{f(\mathbf{i}) : \mathbf{i} \in \widehat{\mathcal{S}}\}$

Theorem: EV$^+$MDDs are canonical

Using saturation to compute the distance function

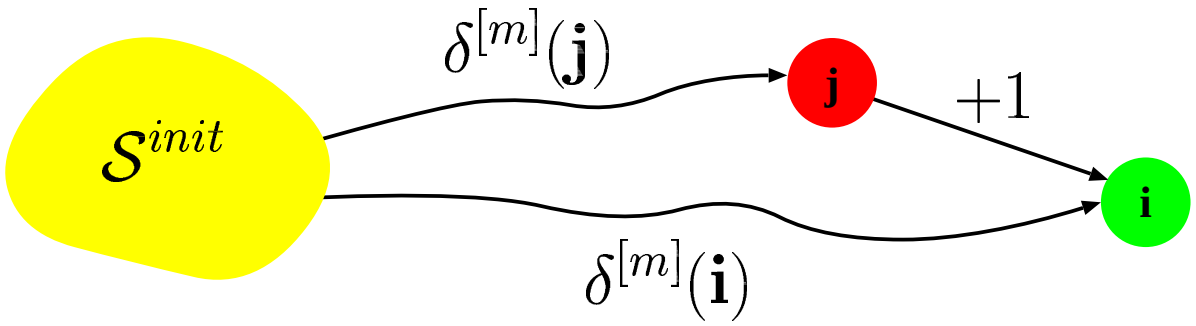
It is easy to build the distance function $\delta : \widehat{\mathcal{S}} \rightarrow \mathbb{N} \cup \{\infty\}$ using a *breadth-first* iteration

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

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

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

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



useful to generate shortest EF witnesses/counterexamples

[work presented at FMCAD 2002]

Results: time and memory to generate and store δ

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

E_s: EV$^+$MDD + saturation E_b: EV$^+$MDD + breadth-first

M_b: multiple MDDs + breadth-first A_s: ADD + saturation A_b: ADD + breadth-first

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 runtimes and peak memory

We developed and implemented better algorithms for symbolic CTL model checking

- exploit locality through a *Kronecker-based encoding of the next-state function*
- employ a *saturation-based algorithm* for EF (easy) and for EU (tricky: *(un)safe events*)
- 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

enormous time and memory improvements for EF and EU

[work presented at CAV 2003]

A successful application of our CTL algorithms: RIPS

A protocol to prevent *incursions* for planes on, approaching, or departing, an airport's runway

Part of the *runway safety monitor* being developed by Lockheed Martin under contract from NASA

3-D real-valued data stored for *ownship* and each *target*: *position* *speed* *acceleration*

Data is used to decide the *status* of each target and trigger an *alarm* for ownship when appropriate

Goal:

- Define a Petri Net model of RIPS
- Use the model checking capabilities of SMART to investigate the potential for *missed alarms*

Approach:

- Use a *discretized view of the monitored zone* around a runway
- Consider *ownship* and one *target* (more targets could be handled)
- Assume *no datalink errors* (so far)

Results:

- Depending on number of targets, size of 3-D grid, speeds: 10^{13} to 10^{42} states
- In spite of our simplifications, we were able to expose flaws in the proposed RIPS protocol
- **The protocol code has been changed due to our findings**

Ongoing work

The SMART package

Initially: SMART: *Simulation and Markovian Analyzer for Reliability and Timing*

- Stochastic models described as Petri nets with random firing times
- Efficient explicit state space generation
- Numerical transient or stationary solution of continuous-time or discrete-time Markov chains
- Batch means simulation of general discrete-state processes

Now: SMART: *Stochastic Model-checking Analyzer for Reliability and Timing*

- Implicit (symbolic, MDD-based) state-space generation and CTL model checking
- Implicit (Kronecker) description of the underlying continuous-time Markov chain
- Numerical stationary solution of a Markov regenerative phase-delay Petri nets (PDPNs)
- Regenerative simulation of general PDPNs, with automatic detection of regenerations

over 100,000 lines of C++ code

Version 1.1 released in Fall 2002, working on a major rewrite

`http://www.cs.ucr.edu/~ciardo/smart`

Explicit local and symbolic global state-space generation

Problem: local state spaces $\mathcal{S}_k$ are not known a priori

Solution: build $\mathcal{S}_k$ "on the fly" (explicitly) alongside the overall state space $\mathcal{S}$ (symbolically)

for each component $\mathbf{i}_k$ of each initial state $(\mathbf{i}_K, \ldots, \mathbf{i}_1) \in \mathcal{S}^{init}$
    $Confirm(\mathbf{i}_k)$;
        (explicitly) build row $\mathbf{N}_{k,\alpha}[\mathbf{i}_k, \cdot]$ for each $\alpha \in \mathcal{E}$ dependent on level k
while the MDD encoding $\mathcal{S}$ has not reached its fixed point w.r.t. $\mathcal{N}$ do
    symbolically explore global states reachable from the currently-known $\mathcal{S}$;
        use the rows $\mathbf{N}_{k,\alpha}[\mathbf{i}_k, \cdot]$ of confirmed local states $\mathbf{i}_k$ only
        confirm $\mathbf{j}_k$ as soon as an entry $\mathbf{N}_{k,\alpha}[\mathbf{i}_k, \mathbf{j}_k]$ is used in a symbolic firing
end while

no need to know a priori the range of each state variable

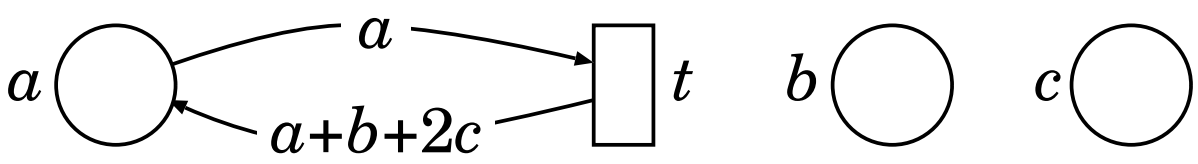
at the end, we have the smallest confi rmed $\mathcal{S}_k$ possible

[work presented at TACAS 2003]

Motivation for 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 causes 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 become too large

[Miner, QEST'04] [joint work with Jinqing Yu]

Variable ordering based on structural information

Variable ordering can have enormous effects on the size of the decision diagram

CUDD dynamically swaps adjacent levels if there is excessive node growth

In our non-binary case, both the *partitioning* and the *ordering* affect the size

We are investigating heuristics to help choose a good ordering:

- idea: minimize the sum of the "spans of events" $\sum_{\alpha \in \mathcal{E}} (Top(\alpha) - Bot(\alpha))$
- An NP-complete problem in itself!
- 5×5 knights swapping game: a speedup of **3** over our best "manual" ordering
- asynchronous circuit verification problem from Barcelona: a speedup of **40**

Partitioning is even harder: we can automatically compute the finest possible partition

- the finest partition is often optimal, but not always
- model *invariants* may suggest good coarsening

[joint work with Radu Siminiceanu]

Conclusions

Symbolic techniques are now very competitive for state-space generation:

- low time and memory requirements for Petri nets especially when using
  - identity-exploiting encodings (Kronecker)
  - locality-exploiting algorithms (saturation)

- generalization are being introduced to cope with
  - local state spaces unknown a priori (nets not covered by invariants)
  - non-Kronecker consistent models (self-modifying nets)

many applications/extensions of these ideas are still unexplored!