What a Structural World

Gianfranco Ciardo

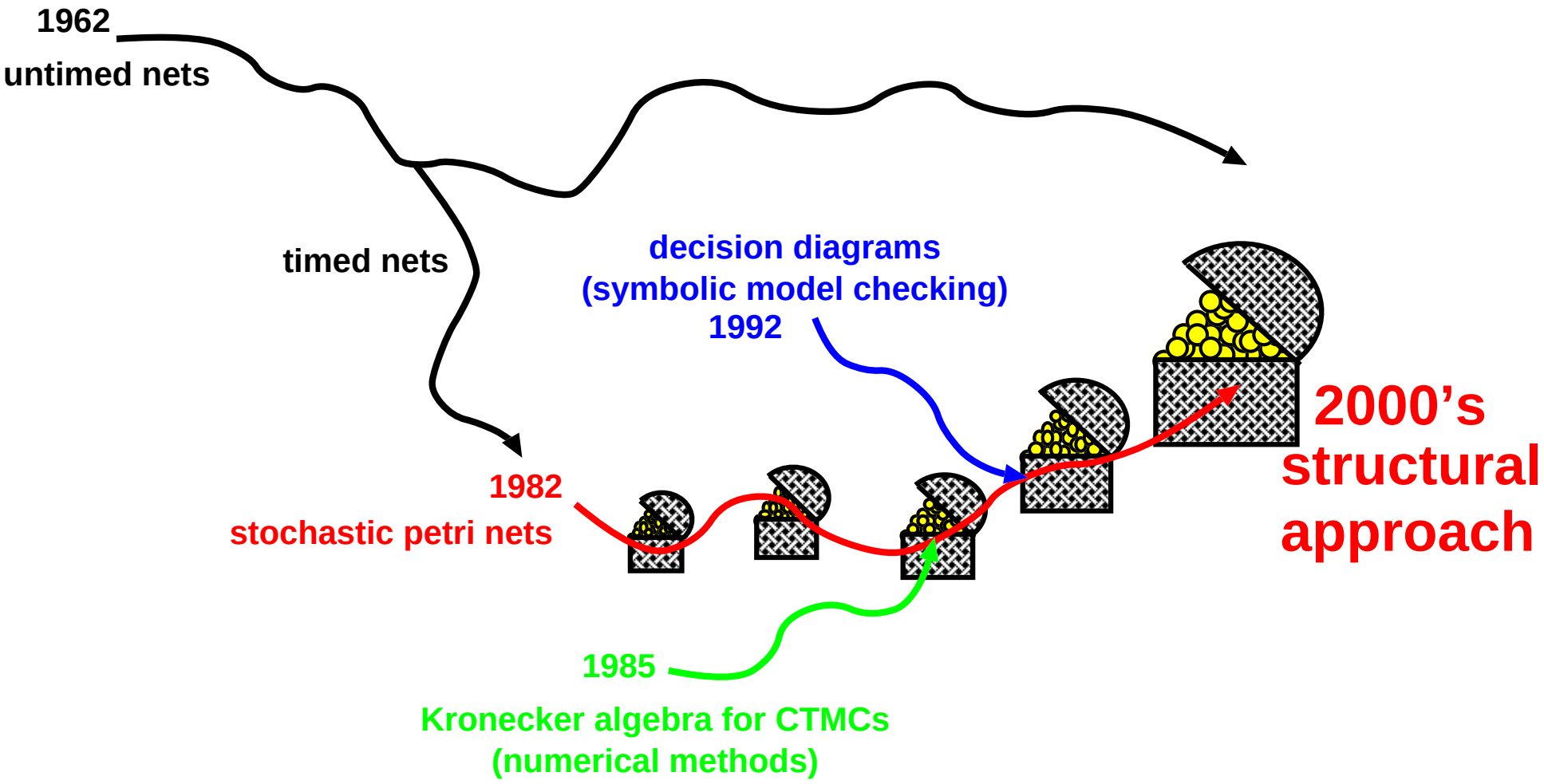
Department of Computer Science, College of William and Mary

Williamsburg, VA 23187, USA

ciardo@cs.wm.edu

Work supported in part by the National Aeronautics and Space Administration (NAG-1-2168).

Map


1962
untimed nets
timed nets
decision diagrams
(symbolic model checking)
1992
1982
stochastic petri nets
2000's
structural
approach
1985
Kronecker algebra for CTMCs
(numerical methods)

Self-modifying stochastic Petri nets with inhibitor arcs

A self-modifying Petri net with inhibitor arcs is a tuple $(\mathcal{P}, \mathcal{T}, I, O, H, \mathbf{s}, \lambda)$ where:

- $\mathcal{P}$ and $\mathcal{T}$ — places and transitions
- $I, O : \mathcal{T} \times \mathcal{P} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N}$ — state-dependent input, output arc cardinalities
- $H : \mathcal{T} \times \mathcal{P} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{N} \cup \{\infty\}$ — state-dependent inhibitor arc cardinalities
- $\mathbf{s} : \mathbb{N}^{|\mathcal{P}|}$ — initial state
- $\lambda : \mathcal{T} \times \mathbb{N}^{|\mathcal{P}|} \to \mathbb{R}^+$ — state-dependent firing rates

Transition t is enabled in a state $\mathbf{i} \in \mathbb{N}^{|\mathcal{P}|}$ iff $\forall p \in \mathcal{P},\ I_{t,p}(\mathbf{i}) \leq \mathbf{i}_p \ \wedge\ H_{t,p}(\mathbf{i}) > \mathbf{i}_p$

If $\mathbf{i} \xrightarrow{t} \mathbf{j}$, the new state $\mathbf{j}$ satisfies $\forall p \in \mathcal{P},\ \mathbf{j}_p = \mathbf{i}_p - I_{t,p}(\mathbf{i}) + O_{t,p}(\mathbf{i})$

The underlying state space $\mathcal{S}$ and transition rate matrix $\mathbf{R}$

We assume a finite state space $\mathcal{S}$

- $\mathbf{s} \in \mathcal{S}$
- $\mathbf{i} \in \mathcal{S} \ \wedge \ \exists t \in \mathcal{T}, \ \mathbf{i} \xrightarrow{t} \mathbf{j} \ \Rightarrow \ \mathbf{j} \in \mathcal{S}$

We assume an ergodic underlying CTMC

- $|\mathcal{S}|$ states, defined by the index mapping

$$\psi : \mathcal{S} \rightarrow \{0, ..., |\mathcal{S}| - 1\}$$

- transition rate matrix $\mathbf{R} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$

$$\mathbf{R}[i, j] = \sum_{t \in \mathcal{T}, \psi(\mathbf{i}) = i, \psi(\mathbf{j}) = j, \mathbf{i} \xrightarrow{t} \mathbf{j}} \lambda_t(\mathbf{i})$$

Fundamental distinction between state $\mathbf{i}$ and its index i

Explicit generation of $\mathcal{S}$ and $\mathbf{R}$

$Explore(\mathcal{P}, \mathcal{T}, I, O, H, \mathbf{s})$ is

1. $\psi(\mathbf{s}) \leftarrow 0$; • *initial state* $\mathbf{s}$ *has index* 0
2. $n \leftarrow 1$; • *number of states so far*
3. $\mathcal{S} \leftarrow \emptyset$; • $\mathcal{S}$ *contains the states explored so far*
4. $\mathcal{U} \leftarrow \{\mathbf{s}\}$; • $\mathcal{U}$ *contains the unexplored states*
5. while $\mathcal{U} \neq \emptyset$ do
6. choose a state $\mathbf{i}$ in $\mathcal{U}$;
7. move $\mathbf{i}$ from $\mathcal{U}$ to $\mathcal{S}$;
8. for each $t \in \mathcal{T}$ do
9. if $I_{t,*}(\mathbf{i}) \leq \mathbf{i} \wedge H_{t,*}(\mathbf{i}) > \mathbf{i}$ then • *test whether* t *is enabled in* $\mathbf{i}$
10. $\mathbf{j} \leftarrow \mathbf{i} - I_{t,*}(\mathbf{i}) + O_{t,*}(\mathbf{i})$; • *build a new state* $\mathbf{j}$ *such that* $\mathbf{i} \xrightarrow{t} \mathbf{j}$
11. if $\mathbf{j} \notin \mathcal{S} \cup \mathcal{U}$ then • *search to determine whether* $\mathbf{j}$ *is a new state*
12. $\psi(\mathbf{j}) \leftarrow n$; • *assign to* $\mathbf{j}$ *the next available index*
13. $n \leftarrow n + 1$;
14. $\mathcal{U} \leftarrow \mathcal{U} \cup \{\mathbf{j}\}$; • *remember to explore* $\mathbf{j}$ *later*
15. end if;
16. $\mathbf{R}[\psi(\mathbf{i}), \psi(\mathbf{j})] \leftarrow \mathbf{R}[\psi(\mathbf{i}), \psi(\mathbf{j})] + \lambda_t(\mathbf{i})$; • *note the indexing!!!*
17. end if;
18. end for;
19. end while;
20. return $\mathcal{S}$, $\mathbf{R}$;

Same old problems

$\mathcal{S}$ is very large, it requires lots of time and memory

$\mathbf{R}$ requires even more memory

Use iterative methods to compute the stationary probability vector $\boldsymbol{\pi} \in \mathbb{R}^{|\mathcal{S}|}$

- **Jacobi** for $j = 0, ..., |\mathcal{S}| - 1$, $\boldsymbol{\pi}^{new}[j] \leftarrow \mathbf{h}[j] \sum_{0 \leq i < |\mathcal{S}|, i \neq j} \boldsymbol{\pi}^{old}[i] \mathbf{R}[i, j]$
- **Gauss-Seidel** for $j = 0, ..., |\mathcal{S}| - 1$, $\boldsymbol{\pi}[j] \leftarrow \mathbf{h}[j] \sum_{0 \leq i < |\mathcal{S}|, i \neq j} \boldsymbol{\pi}[i] \ \mathbf{R}[i, j]$

where $\mathbf{h}[j] = \dfrac{1}{\sum_{0 \leq j < |\mathcal{S}|, j \neq i} \mathbf{R}[i, j]}$ is the expected holding time in state j

20 years of improvements in explicit techniques

- Use hashing or search trees to determine whether a state $\mathbf{j}$ is new (and its index)
- Use sparse storage for each state vector $\mathbf{i}$ (better yet, choose full or sparse dynamically)
- If a bound b on the tokens in place p is known, use $\lceil \log b + 1 \rceil$ bits for $\mathbf{i}_p$
- Use an explicit multilevel data structure to exploit similar sub-states in different states
- Use sparse storage for $\mathbf{R}$
- Use multiple passes to generate and store $\mathbf{R}$:
  1. Generate and store $\mathcal{S}$, while counting (but not storing) the nonzero entries in $\mathbf{R}$
  2. Allocate a memory-efficient data structure to store $\mathbf{R}$ and fill its entries in a second pass
- Use an indexing scheme to store real values (effective if $\mathbf{R}$ has many indentical entries)
- Use secondary storage (disks) to reduce maximum RAM requirements for storing $\mathbf{R}$
- Regenerate the entries of $\mathbf{R}$ on-the-fly at each iteration
- Use numerical methods that require fewer iterations

In a matter of hours, on a workstation with a substantial amount of memory

- ☐ can generate $\mathcal{S}$ with 10^7—10^8 states
- ☐ can solve an underlying CTMC with 10^6—10^7 states and 10^7—10^8 nonzero entries

We need new ideas!

To go further, use implicit methods!!!

decision diagrams

(from symbolic model-checking)

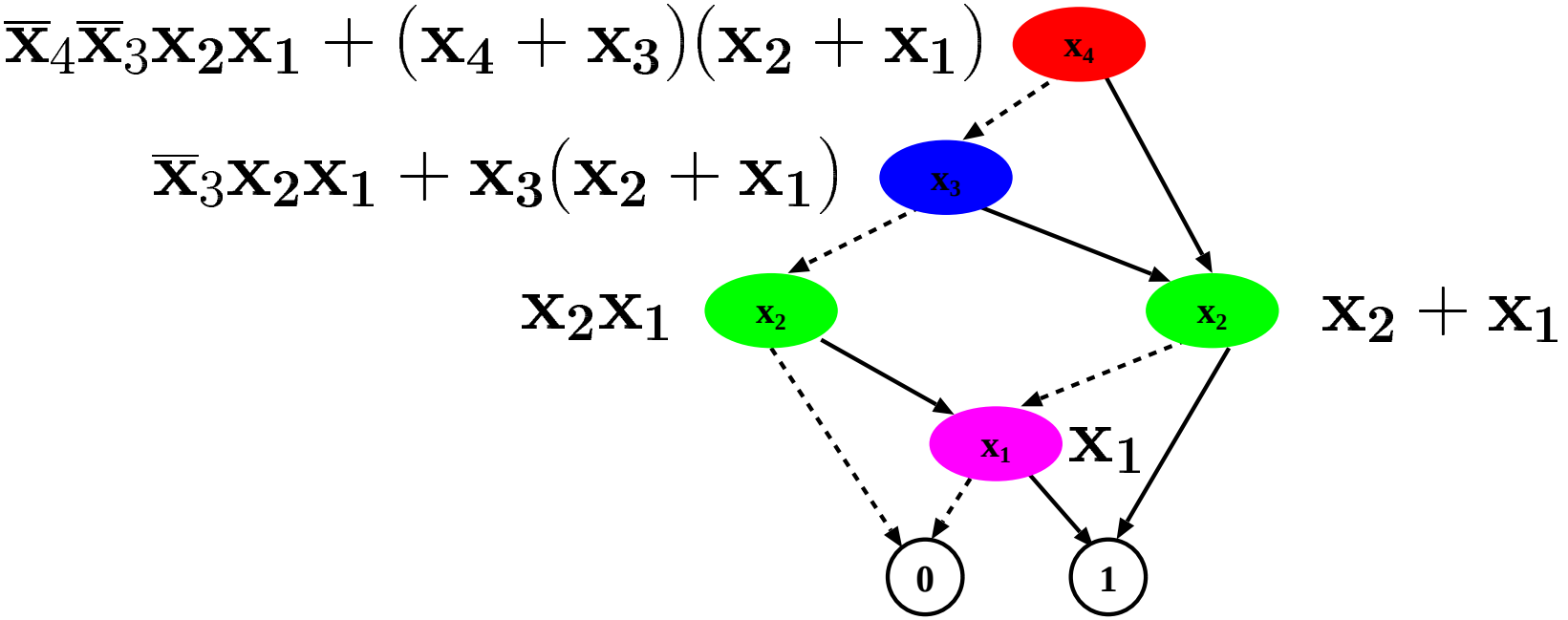
kronecker algebra

(from linear algebra)

(Reduced ordered) binary decision diagrams

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

A canonical representation of boolean functions.



Using BDDs for state space generation

A BDD can encode a subset $\mathcal{S}$ of $\widehat{\mathcal{S}} = \{0,1\}^K$ (think of a safe Petri net with K places)

$(\mathbf{i}_K, ..., \mathbf{i}_1) \in \mathcal{S} \Leftrightarrow$ there is a path from the root to 1 according to $(\mathbf{i}_K, ..., \mathbf{i}_1)$

We can also encode the *next-state function* as a BDD over $2K$ variables

$\mathcal{N} : \widehat{\mathcal{S}} \to 2^{\widehat{\mathcal{S}}}$ defined by $\mathcal{N}(\mathbf{i}) = \{\mathbf{j} : \exists t \in \mathcal{T}, \mathbf{i} \xrightarrow{t} \mathbf{j}\}$

Traditional method

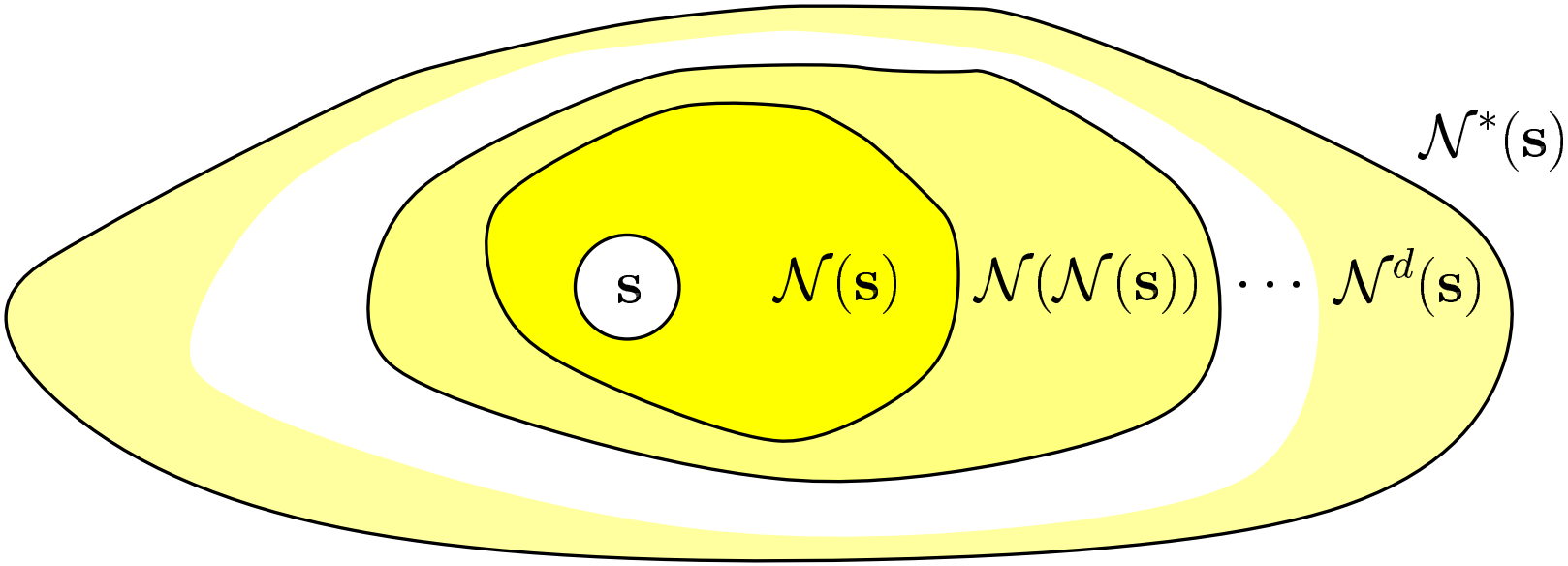
$ExploreBdd(\mathbf{s}, \mathcal{N})$ is
1. $\mathcal{S} \leftarrow \{\mathbf{s}\}$; • *known states*
2. $\mathcal{U} \leftarrow \{\mathbf{s}\}$; • *unexplored states*
3. repeat
4. $\quad \mathcal{X} \leftarrow \mathcal{N}(\mathcal{U})$; • *potentially new states*
5. $\quad \mathcal{U} \leftarrow \mathcal{X} \setminus \mathcal{S}$; • *truly new states*
6. $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{U}$;
7. until $\mathcal{U} = \emptyset$;
8. return $\mathcal{S}$;

Alternative method

$ExploreBdd(\mathbf{s}, \mathcal{N})$ is
1. $\mathcal{S} \leftarrow \{\mathbf{s}\}$;
2. repeat
3. $\quad \mathcal{O} \leftarrow \mathcal{S}$; • *old states*
4. $\quad \mathcal{S} \leftarrow \mathcal{O} \cup \mathcal{N}(\mathcal{O})$; • *new states*
5. until $\mathcal{O} = \mathcal{S}$;
6. return $\mathcal{S}$;

Using BDDs for state space generation (cont.)

The number of iterations is the maximum distance d of any state from the initial state (plus one)



Can study temporal-logic properties of enormous state spaces

"Symbolic model checking: 10^{20} states and beyond"

Burch, Clarke, McMillan, Dill, and Hwang

Kronecker algebra

Given $\mathbf{A} = \begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix}$ $\mathbf{B} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \\ b_{2,0} & b_{2,1} & b_{2,2} \end{bmatrix}$

Kronecker product:

$$\mathbf{A} \otimes \mathbf{B} = \left[\begin{array}{c|c} a_{0,0}\mathbf{B} & a_{0,1}\mathbf{B} \\ \hline a_{1,0}\mathbf{B} & a_{1,1}\mathbf{B} \end{array}\right] = \left[\begin{array}{ccc|ccc} a_{0,0}b_{0,0} & a_{0,0}b_{0,1} & a_{0,0}b_{0,2} & a_{0,1}b_{0,0} & a_{0,1}b_{0,1} & a_{0,1}b_{0,2} \\ a_{0,0}b_{1,0} & a_{0,0}b_{1,1} & a_{0,0}b_{1,2} & a_{0,1}b_{1,0} & a_{0,1}b_{1,1} & a_{0,1}b_{1,2} \\ a_{0,0}b_{2,0} & a_{0,0}b_{2,1} & a_{0,0}b_{2,2} & a_{0,1}b_{2,0} & a_{0,1}b_{2,1} & a_{0,1}b_{2,2} \\ \hline a_{1,0}b_{0,0} & a_{1,0}b_{0,1} & a_{1,0}b_{0,2} & a_{1,1}b_{0,0} & a_{1,1}b_{0,1} & a_{1,1}b_{0,2} \\ a_{1,0}b_{1,0} & a_{1,0}b_{1,1} & a_{1,0}b_{1,2} & a_{1,1}b_{1,0} & a_{1,1}b_{1,1} & a_{1,1}b_{1,2} \\ a_{1,0}b_{2,0} & a_{1,0}b_{2,1} & a_{1,0}b_{2,2} & a_{1,1}b_{2,0} & a_{1,1}b_{2,1} & a_{1,1}b_{2,2} \end{array}\right]$$

Using Kronecker algebra to encode $\mathbf{R}$

- Parallel composition of K SPNs with transitions $\mathcal{T}_k$ and (disjoint) places $\mathcal{P}_k$, $K \geq k \geq 1$
- Overall SPN has transitions $\mathcal{T} = \bigcup_{K \geq k \geq 1} \mathcal{T}_k = \mathcal{T}^S \cup \mathcal{T}^L$ (synchronizing vs. local)
- *Global* state $\mathbf{i}$ is a K-tuple $(\mathbf{i}_K, ..., \mathbf{i}_1)$ of *local* states
- Potential state space is $\widehat{\mathcal{S}} = \mathcal{S}_K \times \cdots \times \mathcal{S}_1 \supseteq \mathcal{S}$
- $\lambda_t(\mathbf{i}) = \lambda_{t,K}(\mathbf{i}_K) \cdots \lambda_{t,1}(\mathbf{i}_1)$ (let $\lambda_{t,k}(\cdot) \equiv 1$ for $t \notin \mathcal{T}_k$)

- $\mathbf{R} = \widehat{\mathbf{R}}[\widehat{\psi}(\mathcal{S}), \widehat{\psi}(\mathcal{S})]$ where $\widehat{\mathbf{R}} = \sum_{t \in \mathcal{T}} \bigotimes_{K \geq k \geq 1} \mathbf{W}_{t,k}$
  - $\mathbf{W}_{t,k}[\mathbf{i}_k, \mathbf{j}_k] = \begin{cases} \lambda_{t,k}(\mathbf{i}_k) & \text{if } \mathbf{i}_k \xrightarrow{t} \mathbf{j}_k \text{ in the } k^{\text{th}} \text{ SPN in isolation} \\ 0 & \text{otherwise} \end{cases}$
  - $\widehat{\psi}(\mathbf{i}) = (\cdots((\mathbf{i}_K n_{K-1} + \mathbf{i}_{K-1}) n_{K-2} \cdots) n_1 + \mathbf{i}_2 n_1) + \mathbf{i}_1$

We encode a huge $\mathbf{R}$ with a few “small” matrices

Using Kronecker algebra to encode $\mathbf{R}$ (cont.)

- To obtain a synchronizing entry of $\mathbf{R}$, multiply K entries in K small matrices
  - Some reuse of common operations is possible $\Rightarrow$ source of overhead
- $\widehat{\mathcal{S}}$ is a superset of $\mathcal{S}$ and $\widehat{\mathbf{R}}$ is a supermatrix of $\mathbf{R}$
  - We can use Power or Jacobi implemented with by-row access...
  - ...$\widehat{\boldsymbol{\pi}}$ of size $|\widehat{\mathcal{S}}|$ to avoid indexing problems... $\Rightarrow$ may waste some memory
  - ...and, at the end of the iterations, $\widehat{\boldsymbol{\pi}}[\widehat{\psi}(\mathbf{i})] > 0 \Leftrightarrow \mathbf{i} \in \mathcal{S}$
  - Or use Gauss-Seidel with by-column access...
  - ...$\boldsymbol{\pi}$ of size $|\mathcal{S}|$ to save memory...
  - ...and an indexing mapping $\psi : \widehat{\mathcal{S}} \rightarrow \{0, ..., |\mathcal{S}| - 1\} \cup \{\text{null}\}$ $\Rightarrow$ additional overhead

One or two orders of magnitude savings in memory

Factor of K overhead in time

"Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models"

Buchholz, Ciardo, Donatelli, and Kemper

A structural approach

Integrate ideas from decision diagrams and Kronecker algebra

- BDDs work best for safe nets but need akward binary encodings for non-safe nets
  our approach uses MDDs to easily manage variables over arbitrary finite sets
- Traditional BDD approach encodes the next-state function $\mathcal{N}$ with a BDD
  our approach uses boolean Kronecker operators for greater efficiency and flexibility
- Traditional BDD approach finds states in breadth-first order
  our approach can explore the firing of transitions in any order
- Traditional Kronecker approach composes “well-behaved” sub-SPNs
  our approach applies to very general SPNs

Closer match between encoding of $\mathcal{S}$ and that of $\mathbf{R}$

Enormous time and memory savings for $\mathcal{S}$ in many models

Substantial efficiency improvements in numerical solution

Structural decomposition of a stochastic Petri net

Given $(\mathcal{P}, \mathcal{T}, I, O, H, \mathbf{s}, \lambda)$ define a consistent partition of its places into $\mathcal{P}_K, \ldots, \mathcal{P}_1$:

$\forall p \in \mathcal{P}_k,\ \forall t \in \mathcal{T},\ \ I_{t,p}, O_{t,p}$, and $H_{t,p}$ can depend only on the local state $\mathbf{i}_k$

For ordinary (non-self-modifying) PNs, any partition will do!

$$\forall \mathbf{i},\ \forall t \in \mathcal{T},\ \ \lambda_t(\mathbf{i}) = \lambda_{t,K}(\mathbf{i}_K) \cdots \lambda_{t,1}(\mathbf{i}_1)$$

Mild restriction (coarsen the partition or split transitions)

Problem:

The local state spaces $\mathcal{S}_k$ are not known a priori, we only know that $\{\mathbf{s}_k\} \subseteq \mathcal{S}_k \subseteq \mathbb{N}^{|\mathcal{P}_k|}$

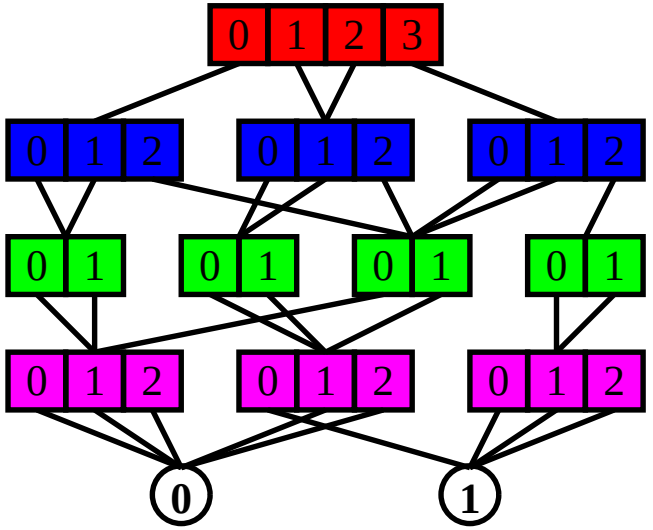
Solution:

We can build $\mathcal{S}_k$ “on the fly” as we build the overall state space $\mathcal{S}$

(Quasi-reduced ordered) multi-valued 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
  - Level 0 contains the only two terminal nodes, $\mathbf{0}$ and $\mathbf{1}$ (false and true).
- For $k > 0$, a node at level k has n_k arcs pointing to nodes at level $k-1$
- No duplicate nodes

$$\mathcal{S}_4 = \{0,1,2,3\}$$
$$\mathcal{S}_3 = \{0,1,2\}$$
$$\mathcal{S}_2 = \{0,1\}$$
$$\mathcal{S}_1 = \{0,1,2\}$$

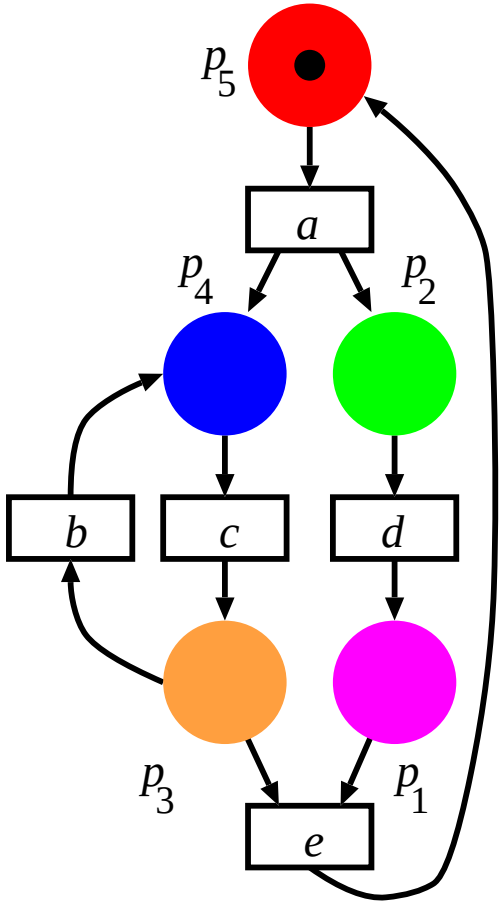


$$\mathcal{S} = \left\{ \begin{array}{cccccccccccccccccc} 0 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 0 & 0 & 1 & 1 & 2 & 0 & 0 & 1 & 1 & 2 & 0 & 1 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 &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

Encoding the next-state function (example, $K = 5$)

$\mathcal{S}_5 = \{0,1\}$ $\mathcal{S}_4 = \{0,1\}$ $\mathcal{S}_3 = \{0,1\}$ $\mathcal{S}_2 = \{0,1\}$ $\mathcal{S}_1 = \{0,1\}$

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

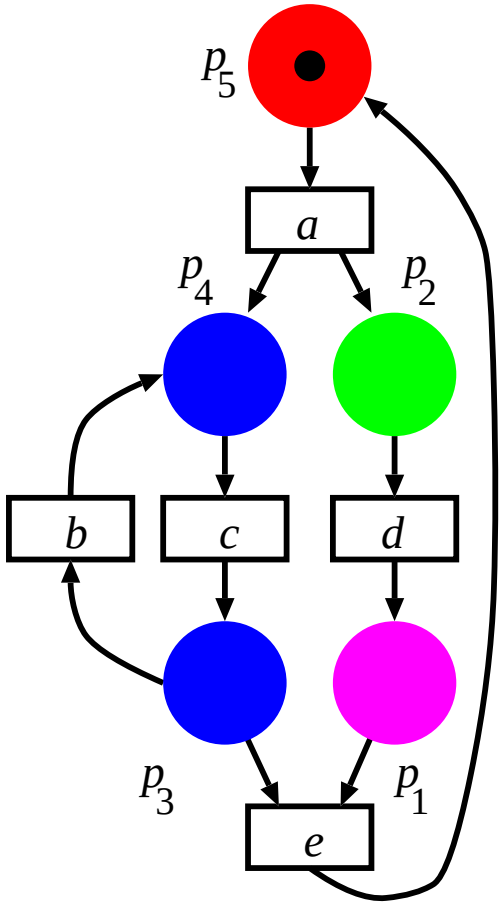


Encoding the next-state function (example, $K = 4$)

$First(b) = Last(b) = First(c) = Last(c) = 3$: merge b and c into a single **local** event l

$\mathcal{S}_4 = \{0,1\}$ $\mathcal{S}_3 = \{(0p_4, 0p_3), (1p_4, 0p_3), (0p_4, 1p_3)\} = \{0,1,2\}$ $\mathcal{S}_2 = \{0,1\}$ $\mathcal{S}_1 = \{0,1\}$

$\mathcal{N}_{4,a} : \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$			$\mathcal{N}_{4,e} : \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$
$\mathcal{N}_{3,a} : \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\mathcal{N}_{3,l} : \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$		$\mathcal{N}_{3,e} : \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$
$\mathcal{N}_{2,a} : \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$		$\mathcal{N}_{2,d} : \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$	
		$\mathcal{N}_{1,d} : \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$	$\mathcal{N}_{1,e} : \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$
$First : 4$ $Last : 2$	$First : 3$ $Last : 3$	$First : 2$ $Last : 1$	$First : 4$ $Last : 1$



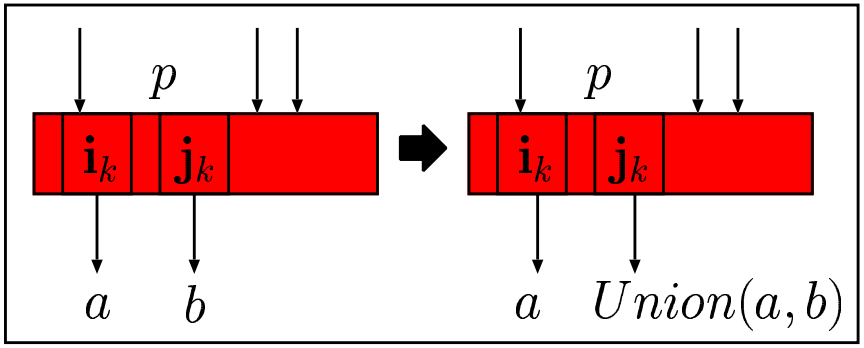
Locality, symmetry, and monotonicity in transition firing

Decompose the next-state function: $\mathcal{N}(\mathbf{i}) = \bigcup_{t\in\mathcal{T}} \mathcal{N}_t(\mathbf{i}) = \bigcup_{t\in\mathcal{T}} \mathcal{N}_{t,K}(\mathbf{i}_K) \times \cdots \times \mathcal{N}_{t,1}(\mathbf{i}_1)$

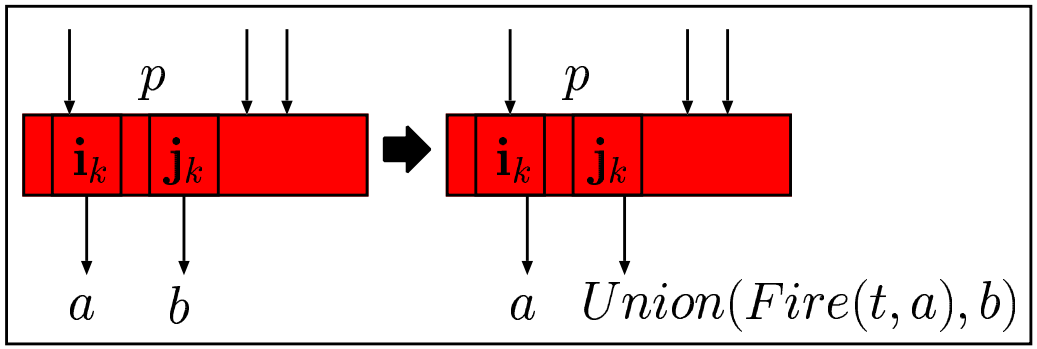
If $\mathbf{i} \in \mathcal{S}$, $\mathbf{i} \xrightarrow{t} \mathbf{j}$, $First(t) = k \wedge Last(t) = l$, then $\mathbf{j} = (\mathbf{i}_K, \ldots, \mathbf{i}_{k+1}, \mathbf{j}_k, \ldots, \mathbf{j}_l, \mathbf{i}_{l-1}, \ldots, \mathbf{i}_1)$

If also $\mathbf{i}' \in \mathcal{S}$, $(\mathbf{i}_k, \ldots, \mathbf{i}_1) = (\mathbf{i}'_k, \ldots, \mathbf{i}'_1)$, then $\mathbf{i}' \xrightarrow{t} \mathbf{j}'$, where $\mathbf{j}' = (\mathbf{i}'_K, \ldots, \mathbf{i}'_{k+1}, \mathbf{j}_k, \ldots, \mathbf{j}_1)$

Local transitions



Syncronizing transitions



This can save huge amounts of computation

Saturation: an efficient iteration strategy

One traditional application of a monolithic $\mathcal{N}$ is equivalent to

$$\begin{aligned} \mathcal{X}^{(t_1)} &\leftarrow \mathcal{N}_{t_1}(\mathcal{S}); \\ &\dots \\ \mathcal{X}^{(t_{|\mathcal{T}|})} &\leftarrow \mathcal{N}_{t_{|\mathcal{T}|}}(\mathcal{S}); \\ \mathcal{S} &\leftarrow \mathcal{S} \cup \mathcal{X}^{(t_1)} \cup \ldots \cup \mathcal{X}^{(t_{|\mathcal{T}|})}; \end{aligned}$$

We can speed up by doing

$$\begin{aligned} \mathcal{S} &\leftarrow \mathcal{S} \cup \mathcal{N}_{t_1}(\mathcal{S}); \\ &\dots \\ \mathcal{S} &\leftarrow \mathcal{S} \cup \mathcal{N}_{t_{|\mathcal{T}|}}(\mathcal{S}); \end{aligned}$$

And even better yet by doing

$$\begin{aligned} \mathcal{S} &\leftarrow \mathcal{S} \cup \mathcal{N}^*_{t_1}(\mathcal{S}); \\ &\dots \\ \mathcal{S} &\leftarrow \mathcal{S} \cup \mathcal{N}^*_{t_{|\mathcal{T}|}}(\mathcal{S}); \end{aligned}$$

By far the best improvement: **saturate** each MDD node recursively starting from the bottom node

(a node at level k is **saturated** if it is a fixed point w.r.t. all transitions t s.t. $First(t) = k$)

Enormous savings in both time and (peak) memory

Results for state space generation

(1 Ghz Pentium)	Model Size	Size of $\mathcal{S}$	time (sec.)	final memory	peak memory
Dining Phils	1000	9.18×10^{626}	4.05	134 Kb	159 Kb
	2000	8.43×10^{1253}	15.47	268 Kb	318 Kb
	3000	7.74×10^{1880}	38.89	402 Kb	477 Kb
Slotted Ring	30	1.03×10^{31}	2.52	18 Kb	243 Kb
	60	7.29×10^{62}	18.28	74 Kb	1.7 Mb
	100	2.60×10^{105}	82.23	203 Kb	7.4 Mb
Round Robin	100	2.85×10^{32}	3.65	189 Kb	199 Kb
	200	7.23×10^{62}	30.34	729 Kb	747 Kb
	300	1.37×10^{94}	127.72	1.6 Mb	1.6 Mb
FMS	75	6.98×10^{19}	5.45	445 Kb	910 Kb
	100	2.70×10^{21}	12.16	858 Kb	1.8 Mb
	200	1.95×10^{25}	89.56	4.7 Mb	11 Mb
Kanban	25	7.68×10^{12}	10.45	37 Kb	848 Kb
	50	1.04×10^{16}	206.28	409 Kb	19.1 Mb
	75	7.83×10^{17}	1314.50	1.8 Mb	128.8 Mb

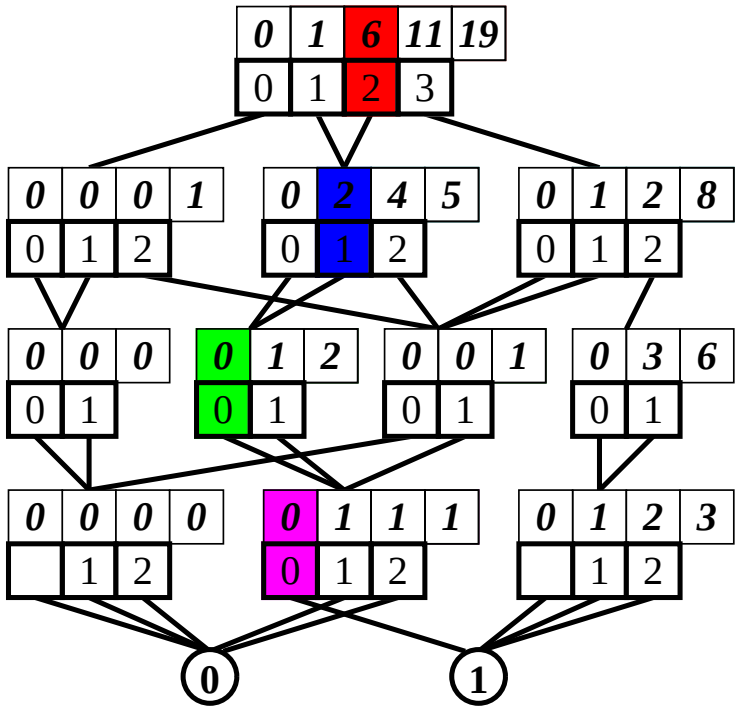
Structural solution of the CTMC

We want to use π of size $|\mathcal{S}|$, not $\hat{\pi}$ of size $|\hat{\mathcal{S}}|$

We want to use Gauss-Seidel/SOR, not Power/Jacobi

And we don't want to pay much for this

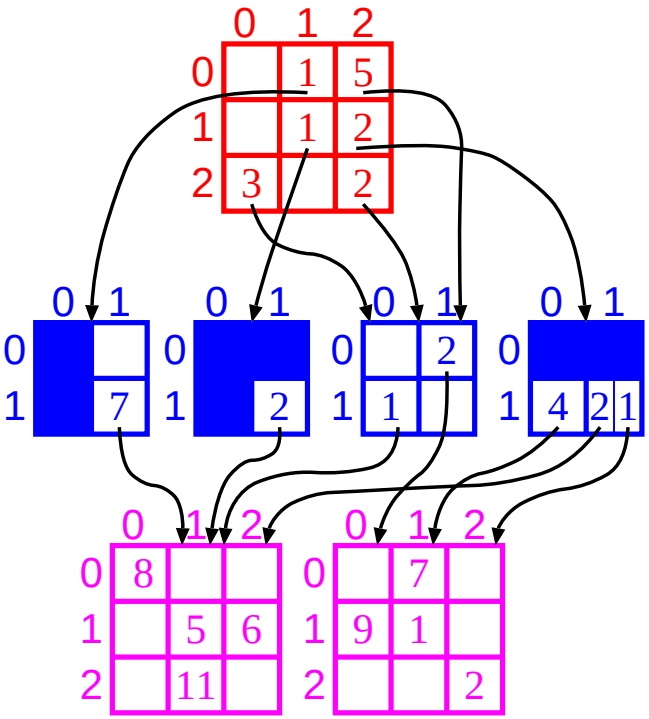
We must be able to compute $\psi : \hat{\mathcal{S}} \rightarrow \{0, ..., |\mathcal{S}| - 1\}$ efficiently



$$\psi(2,1,0,0) = 6 + 2 + 0 + 0 = 8$$

Matrix diagrams

A way to encode both $\mathcal{S}$ and $\mathbf{R}$ exactly (not $\widehat{\mathcal{S}}$ and $\widehat{\mathbf{R}}$, their potential larger versions)



	0 0 0	0 0 0	0 0 0	1 1 1	1 1 1	1 1 1	2 2 2	2 2 2	2 2 2
	0 0 0	1 1 1	2 2 2	0 0 0	1 1 1	2 2 2	0 0 0	1 1 1	2 2 2
	0 1 2	0 1 2	0 1 2	0 1 2	0 1 2	0 1 2	0 1 2	0 1 2	0 1 2
0 0 0								70	
0 0 1								90 10	
0 0 2									20
0 1 0						56	40		
0 1 1						35 42	25 30		
0 1 2						77	55		
1 0 0									
1 0 1									
1 0 2									
1 1 0						16	56	32 14	
1 1 1						10 12	72 8	18 22	24
1 1 2						22	16	44	4
2 0 0				42				28	
2 0 1			54 6					36 4	
2 0 2					12				8
2 1 0	24						16		
2 1 1		15 18					10 12		
2 1 2		33					22		

Results for the numerical solution

Time requirements for the Kanban model (450MHz Pentium workstation with 384 Mbytes RAM)

N	$\lvert\mathcal{S}\rvert$	number of arcs in $\mathbf{R}$	Matr. diagr.		Kronecker				Explicit	
			Gauss-Seidel		Gauss-Seidel		Jacobi		Gauss-Seidel	
			Iters	sec/iter	Iters	sec/iter	Iters	sec/iter	Iters	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	—	—	—	—

What next?

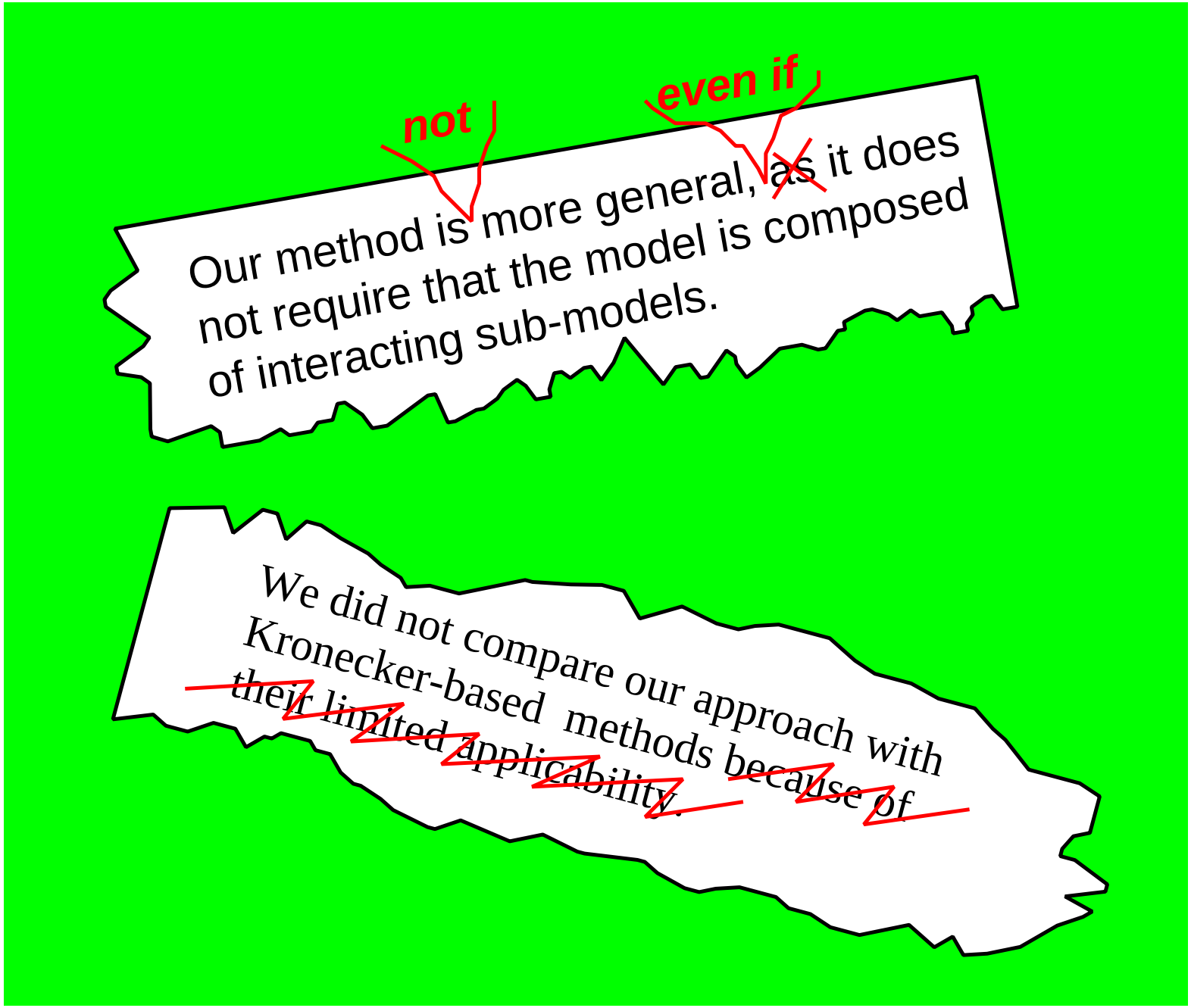
Structural solution approaches are here to stay...

... but much work remains to be done

- **Generalized Kronecker products** might overcome some limitations in applicability
- **Probabilistic decision diagrams** and **canonical matrix diagrams** should be investigated
- An efficient **implicit encoding of the probability vector** is now the holy grail
- If it cannot be found, we need to use structural-based **approximations**
- Good **heuristics** for choosing a partition and ordering the levels
- Go out in the real world and use these techniques: they are ready for prime time!!!

Exhibits A, B (fake, but they might as well be real)

Exhibits A, B (corrected)



What a Structural World

(with apologies to Louis Armstrong)

I see process algebras . . . Petri nets too

I watch their state spaces . . . exponentially accrue

And I think to myself . . . what a structural world.