Modeling the Logical Behavior of Discrete–state Systems

Gianfranco Ciardo

Department of Computer Science

College of William and Mary

Williamsburg, VA 23187, USA

ciardo@cs.wm.edu

Question

What is a BDD (Boolean Decision Diagram)?

What is the most cited computer science document in citeseer?
(http://citeseer.nj.nec.com/)

Answer

Most cited source documents in the ResearchIndex database as of November 2001

This list only includes documents in the ResearchIndex database. Citations where one or more authors of the citing and cited articles match are not included. The data is automatically generated and may contain errors. The list is generated in batch mode and citation counts may differ from those currently in the ResearchIndex database, because the database is continuously updated.

1. Graph-Based Algorithms for Boolean Function Manipulation - Bryant (1986) (Correct)
In this paper we present a new data structure for representing Boolean functions and an associated set of...

2. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems - Rivest, Shamir, Adleman (1978) (Correct)
An encryption method is presented with the novel property that publicly revealing an encryption key does not...

Discrete-state vs. continuous-state systems

At any instant of time, a *system* is in a given *state* $\mathbf{i}$

The set $\mathcal{S}$ of possible states is called *state-space*

Example of *continuous state-spaces*:

- Weight of an organism: $\mathcal{S} = [0, +\infty)$
- Level of liquid in a tank: $\mathcal{S} = [0, maxlevel]$

Example of *discrete state-spaces*:

- Number of passengers in an airplane: $\mathcal{S} = \{0, 1, ..., maxseats\}$
- Number and type of available airplanes: $\mathcal{S} = \{(n_1, ..., n_t) : t \in \mathbb{N}, n_1, ..., n_t \in \mathbb{N}\}$
- Value of the program counter and of each variable in a running computer program

we consider only discrete-state systems

The structure of the state

global state (of the entire system) vs. *local* state (of a subsystem)

The (global) state is a collection of the local state of each subsystem

For example, consider a program with

- Program counter variable p
- Boolean variables $b_1, ..., b_B$
- Integer variables $i_1, ..., i_I$
- Real variables $r_1, ..., r_R$

Each variable corresponds to a local state

Their union corresponds to the (global) state of the program:

$$\mathcal{S} \subseteq \{0, ..., maxpc\} \times \{0, 1\}^B \times \{minint, ..., maxint\}^I \times \{minreal, ..., maxreal\}^R$$

think of a state as a (boolean or integer) vector

Discrete-state models

A *discrete state model* is fully specified by:

- *potential state space* $\widehat{\mathcal{S}}$ (the "type" of the state)
- *initial state* $\mathbf{s}^{init} \in \widehat{\mathcal{S}}$ sometimes we have a set of initial states, $\mathcal{S}^{init}$
- *next-state function* $\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}}$ naturally extended to sets: $\mathcal{N}(\mathcal{X}) = \bigcup_{\mathbf{i}\in\mathcal{X}} \mathcal{N}(\mathbf{i})$

The *state space* $\mathcal{S}$ of the model, assumed finite, is the smallest set satisfying:

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

State $\mathbf{i}$ is a *trap* or *absorbing* if $\mathcal{N}(\mathbf{i}) = \emptyset$

We often define a set $\mathcal{E}$ of model *events* and decompose $\mathcal{N}(\mathbf{i}) = \bigcup_{e\in\mathcal{E}} \mathcal{N}_e(\mathbf{i})$

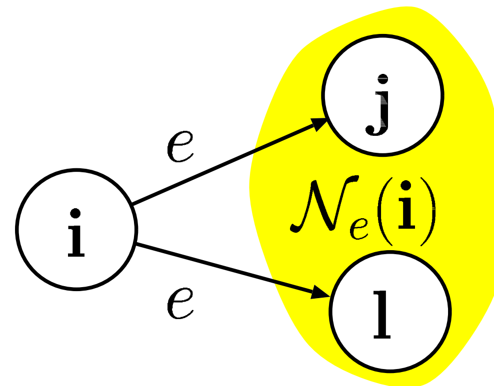
Event e is *disabled* in state $\mathbf{i}$ if $\mathcal{N}_e(\mathbf{i}) = \emptyset$

Event e is *enabled* in state $\mathbf{i}$ if $\mathcal{N}_e(\mathbf{i}) \neq \emptyset$, we write $\mathbf{i}\xrightarrow{e}$ or $e \in \mathcal{E}(\mathbf{i})$

If $\mathbf{j} \in \mathcal{N}_e(\mathbf{i})$, we write $\mathbf{i}\xrightarrow{e}\mathbf{j}$

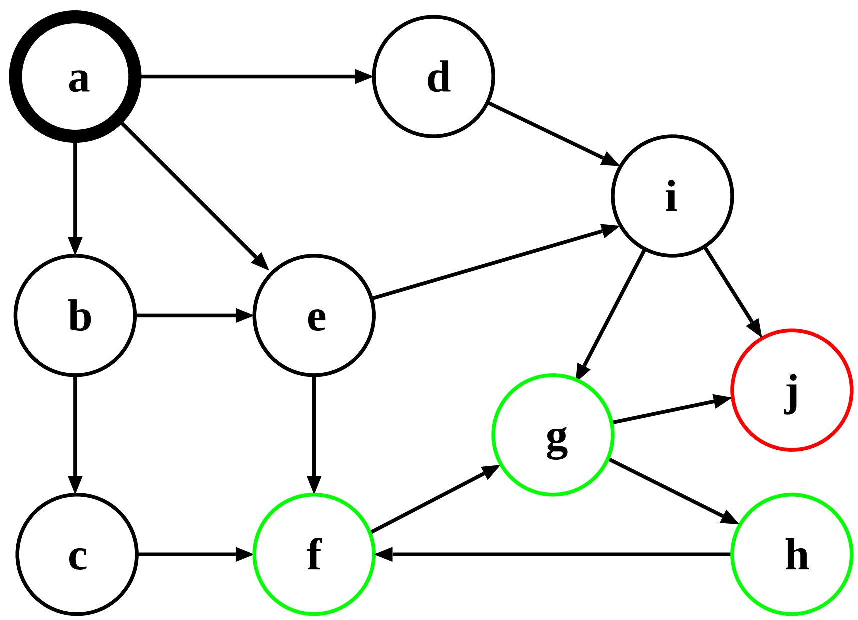
Examples of next-state function and state space

$\mathcal{N}_e(\mathbf{i})$ is the set of states that can nondeterministically be reached from $\mathbf{i}$ when e occurs (or fires)



If $\mathcal{N}_e(\mathbf{i}) = \emptyset$, e is *disabled* in $\mathbf{i}$, otherwise it is *enabled*

An example of state space with one absorbing state and one recurrent class



Petri nets

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

- $\mathcal{P}$ set of *places*, drawn as circles
- $\mathcal{E}$ set of *events*, or *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
- $\mathbf{s}^{init} \in \mathbb{N}^{|\mathcal{P}|}$ initial *state*, or *marking*

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

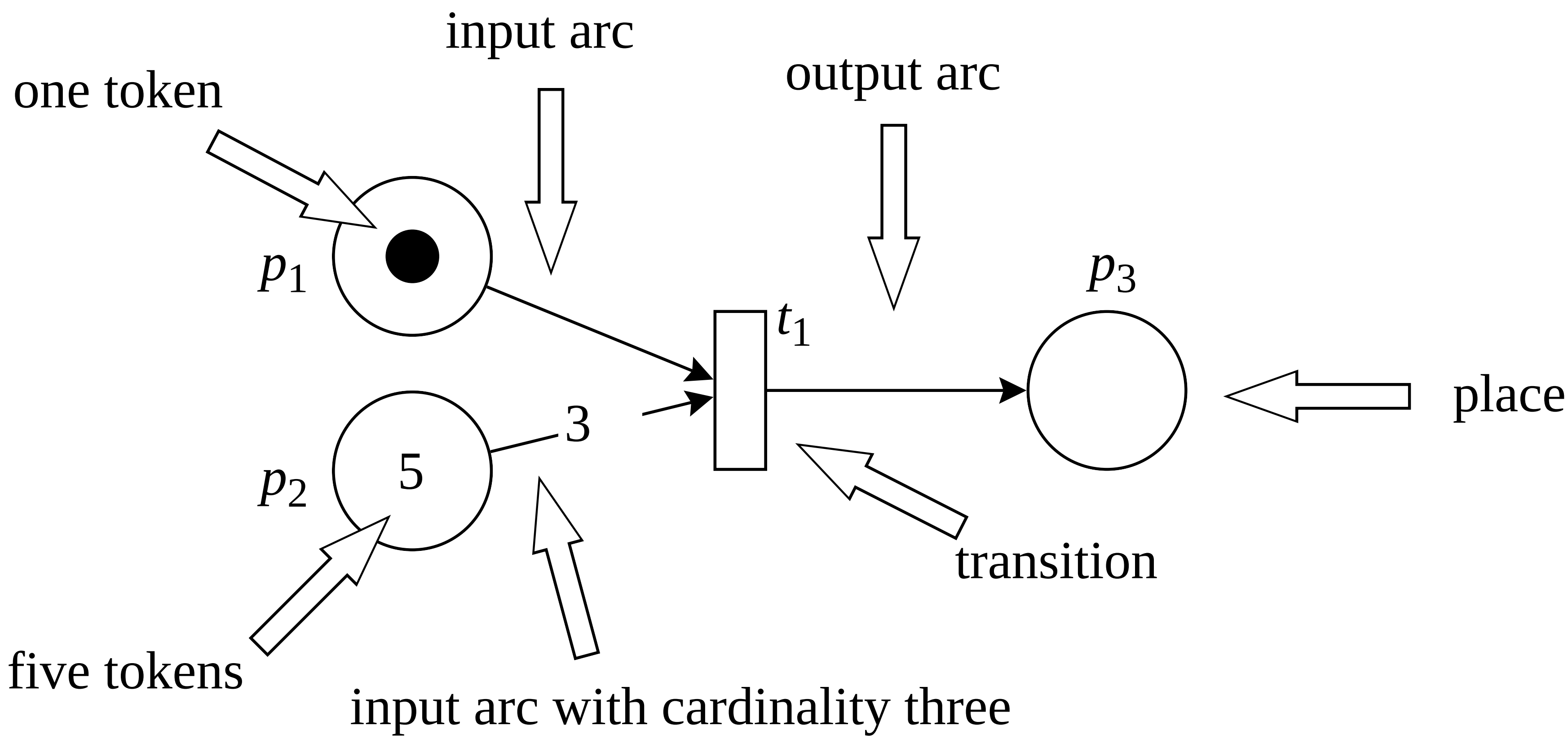
Condition for event e to be *enabled* in state $\mathbf{i} \in \mathbb{N}^{|\mathcal{P}|}$: $e \in \mathcal{E}(\mathbf{i}) \;\Leftrightarrow\; \forall p \in \mathcal{P},\; \mathbf{D}^-_{p,e} \le \mathbf{i}_p$

An event e enabled in state $\mathbf{i}$ can *fire*: $\mathbf{i} \xrightarrow{e} \mathbf{j} \;\Leftrightarrow\; \forall p \in \mathcal{P},\; \mathbf{j}_p = \mathbf{i}_p - \mathbf{D}^-_{p,e} + \mathbf{D}^+_{p,e}$

Thus, $\mathbf{j} \in \mathcal{N}(\mathbf{i}) \;\Leftrightarrow\; \exists e \in \mathcal{E},\; \mathbf{i} \xrightarrow{e} \mathbf{j}$

The *state space*, or *reachability set*, $\mathcal{S}$ is defined as usual

Graphical representation of a Petri net



[redacted]

Enabling rule

$e \in \mathcal{E}(\mathbf{i})$ iff each input arc contains at least as many tokens as the cardinality of the input arc:

$$\forall p \in \mathcal{P},\ \mathbf{D}^{-}_{p,e} \leq \mathbf{i}_p \quad \text{or, in vector form} \quad \mathbf{D}^{-}_{\bullet,e} \leq \mathbf{i}$$

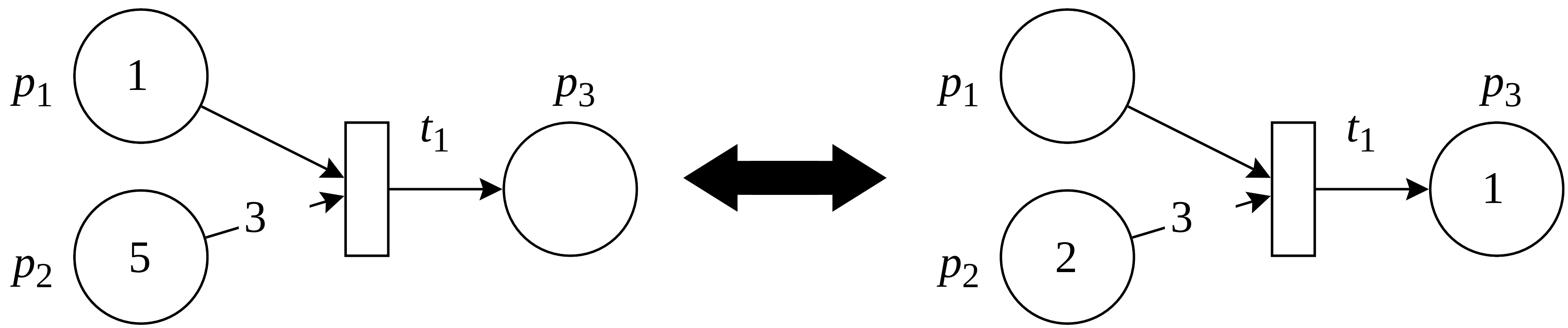
Firing rule

If $\mathbf{i} \xrightarrow{e} \mathbf{j}$, we obtain $\mathbf{j}$ by removing tokens from input places and adding tokens to output places:

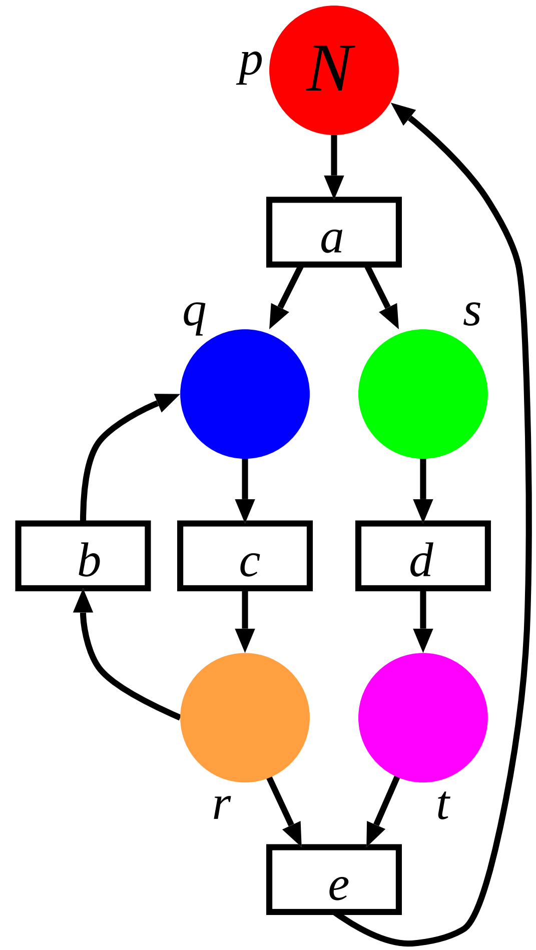
$$\forall p \in \mathcal{P},\ \mathbf{j}_p = \mathbf{i}_p - \mathbf{D}^{-}_{p,e} + \mathbf{D}^{+}_{p,e} \quad \text{or, in vector form} \quad \mathbf{j} = \mathbf{i} - \mathbf{D}^{-}_{\bullet,e} + \mathbf{D}^{+}_{\bullet,e} = \mathbf{i} + \mathbf{D}_{\bullet,e}$$

where $\mathbf{D} = \mathbf{D}^{+} - \mathbf{D}^{-}$ is the *incidence matrix*

For example, if t_1 fires:



[redacted]

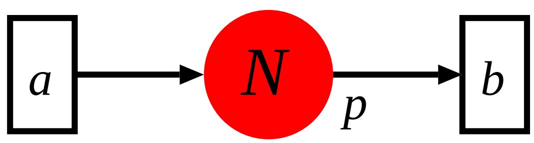


If the initial state is $\mathbf{s}^{init} = ({\color{red}N}, {\color{blue}0}, {\color{orange}0}, {\color{green}0}, {\color{magenta}0})$:

$\mathcal{S}$ contains $\dfrac{(N+1)(N+2)(2N+3)}{6}$ reachable states

For any initial state $\mathbf{s}^{init} = ({\color{red}N})$:

$\mathcal{S}$ contains ∞ reachable states



State-by-state generation of $\mathcal{S}$

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

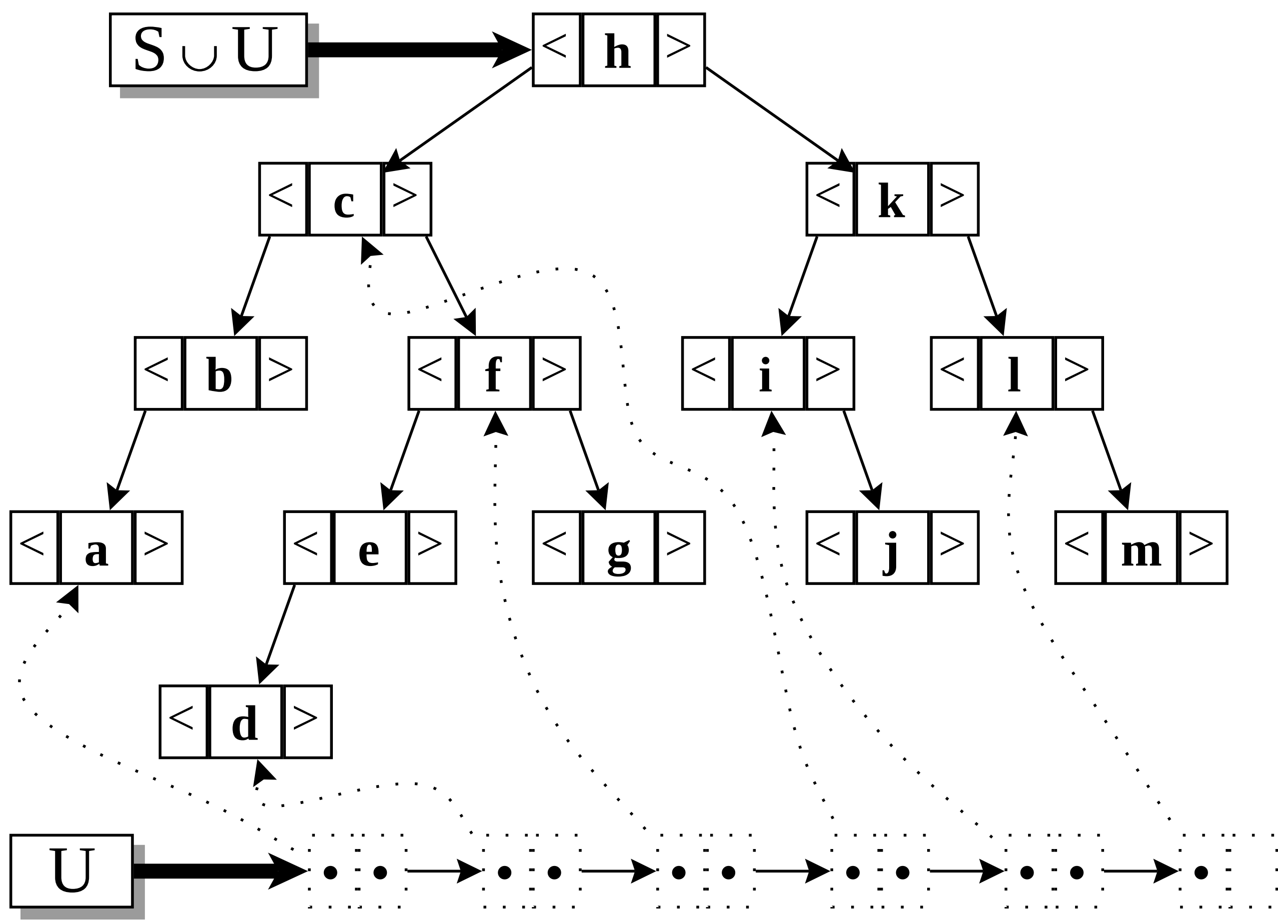
```
 1. S ← {s^init};                                      • S contains the states known so far
 2. U ← {s^init};                                      • U contains the unexplored states known so far
 3. while U ≠ ∅ do
 4.     choose a state i in U and remove it from U;
 5.     for each j ∈ N(i) do
 6.         if j ∉ S then                              • search to determine whether j is a new state
 7.             S ← S ∪ {j};
 8.             U ← U ∪ {j};                           • remember to explore j later
 9.         end if;
10.     end for;
11. end while;
12. return S;
```

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

How can we store $\mathcal{S}$ and $\mathcal{U}$ efficiently?

If we store $\mathcal{S}$ and $\mathcal{U}$ together, we can distinguish them using a linked list for $\mathcal{U}$

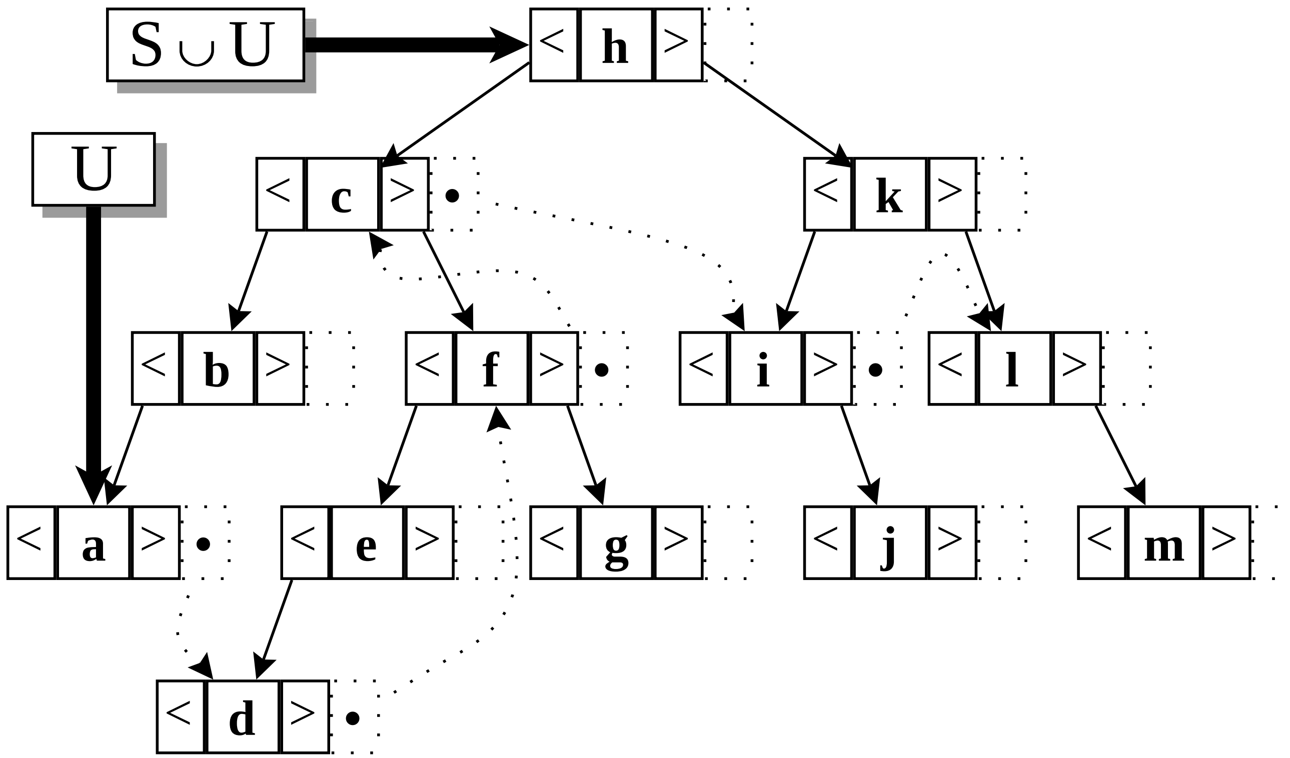
$\Rightarrow$ Additional $2 \cdot |\mathcal{U}| \cdot B_{pointer}$ bits



[redacted]

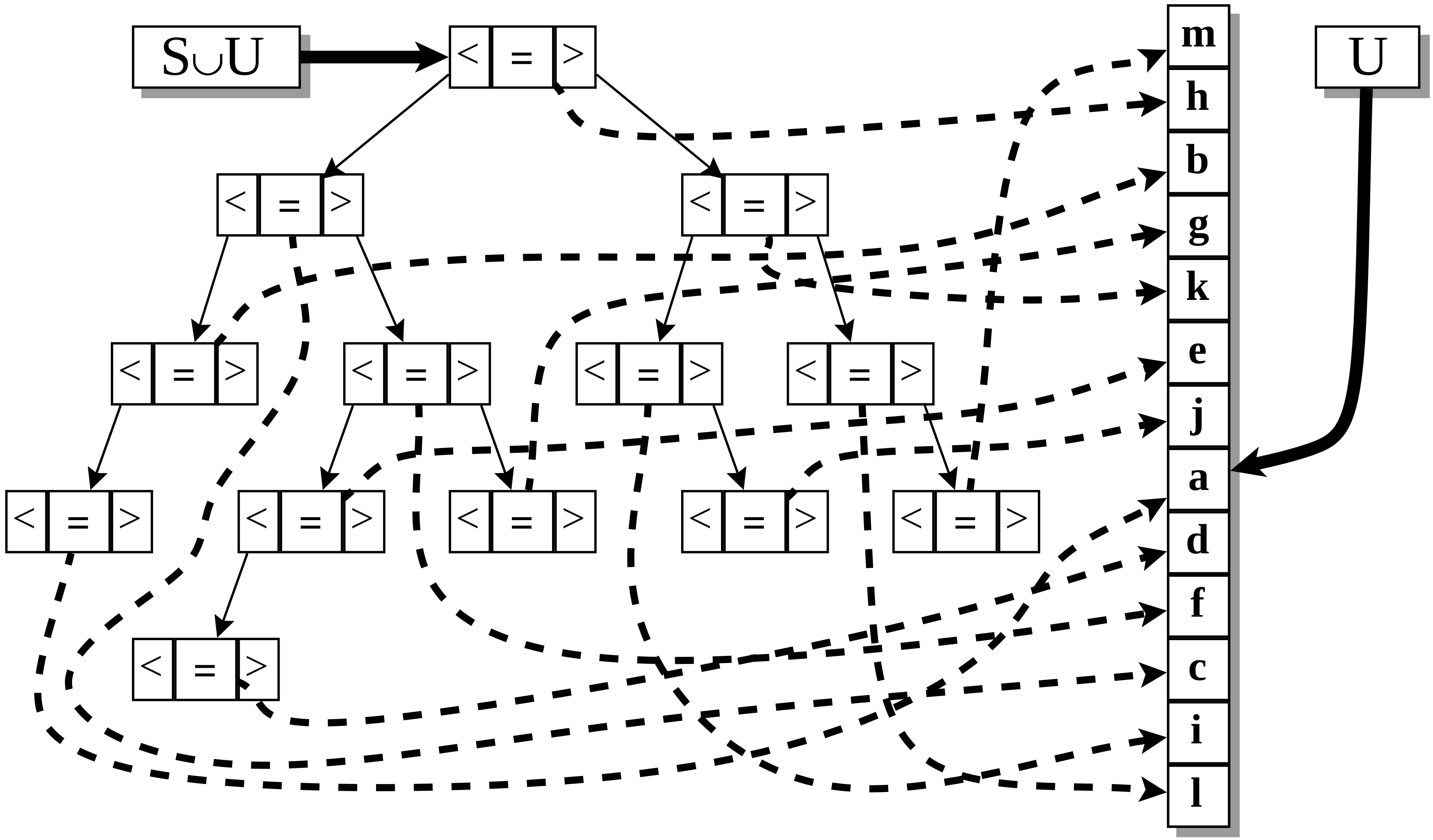
Or a pointer the next unexplored state, in each tree node

$\Rightarrow$ Additional $|\mathcal{S}| \cdot B_{pointer}$ bits

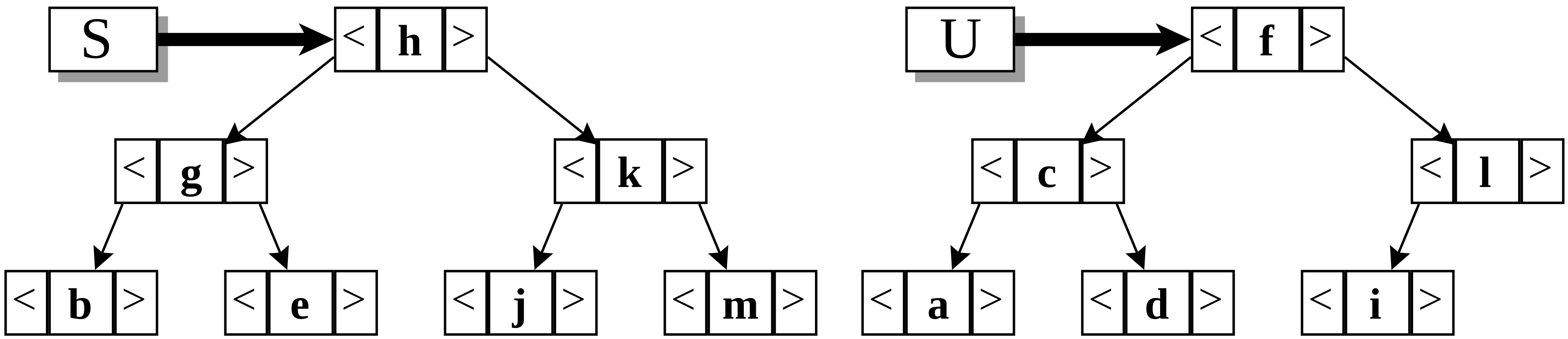


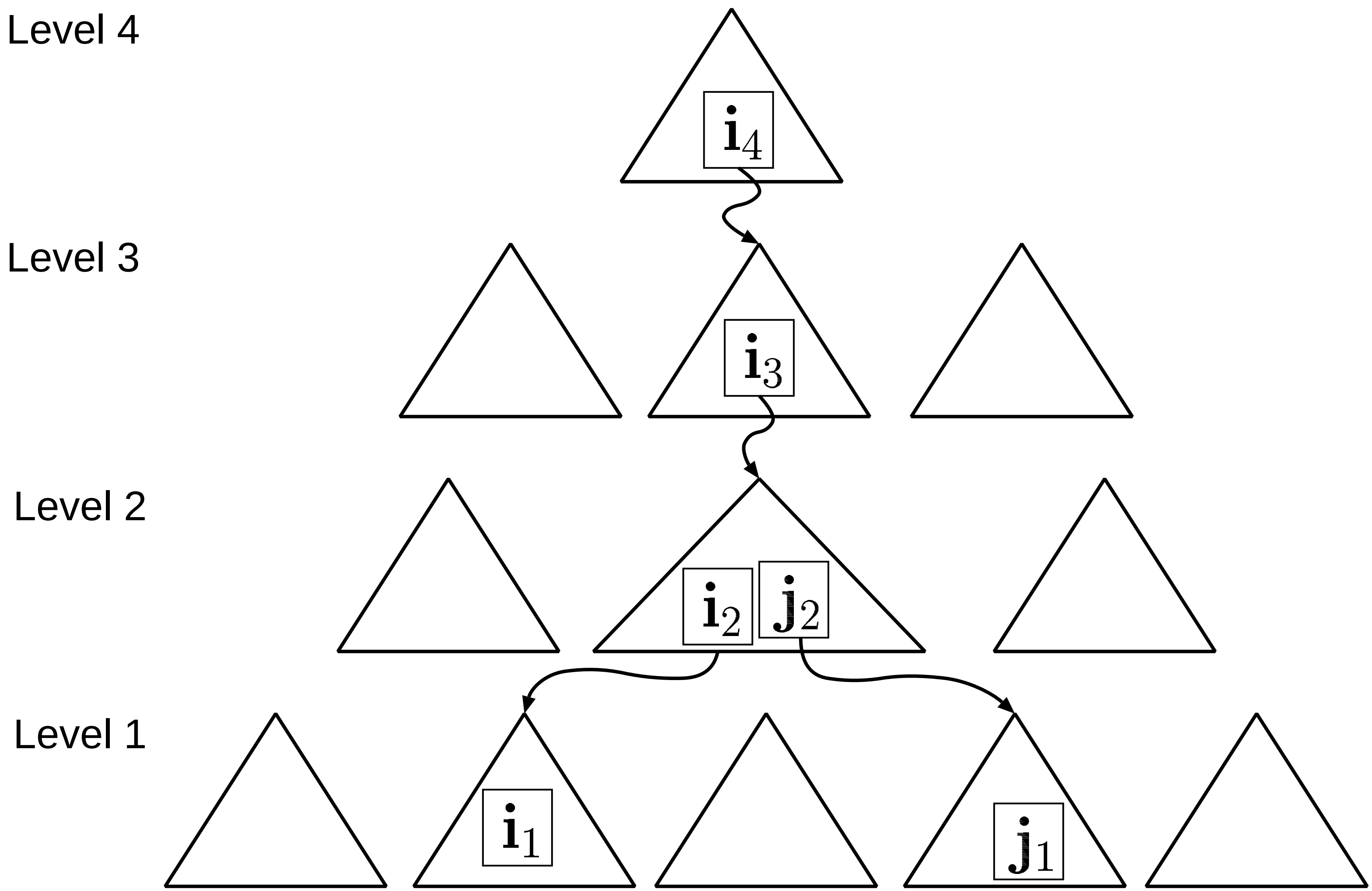
Or store the states in a dynamic array structure

$\Rightarrow$ Additional $|\mathcal{S}| \cdot B_{index}$ bits



If we store $\mathcal{S}$ and $\mathcal{U}$ separately:

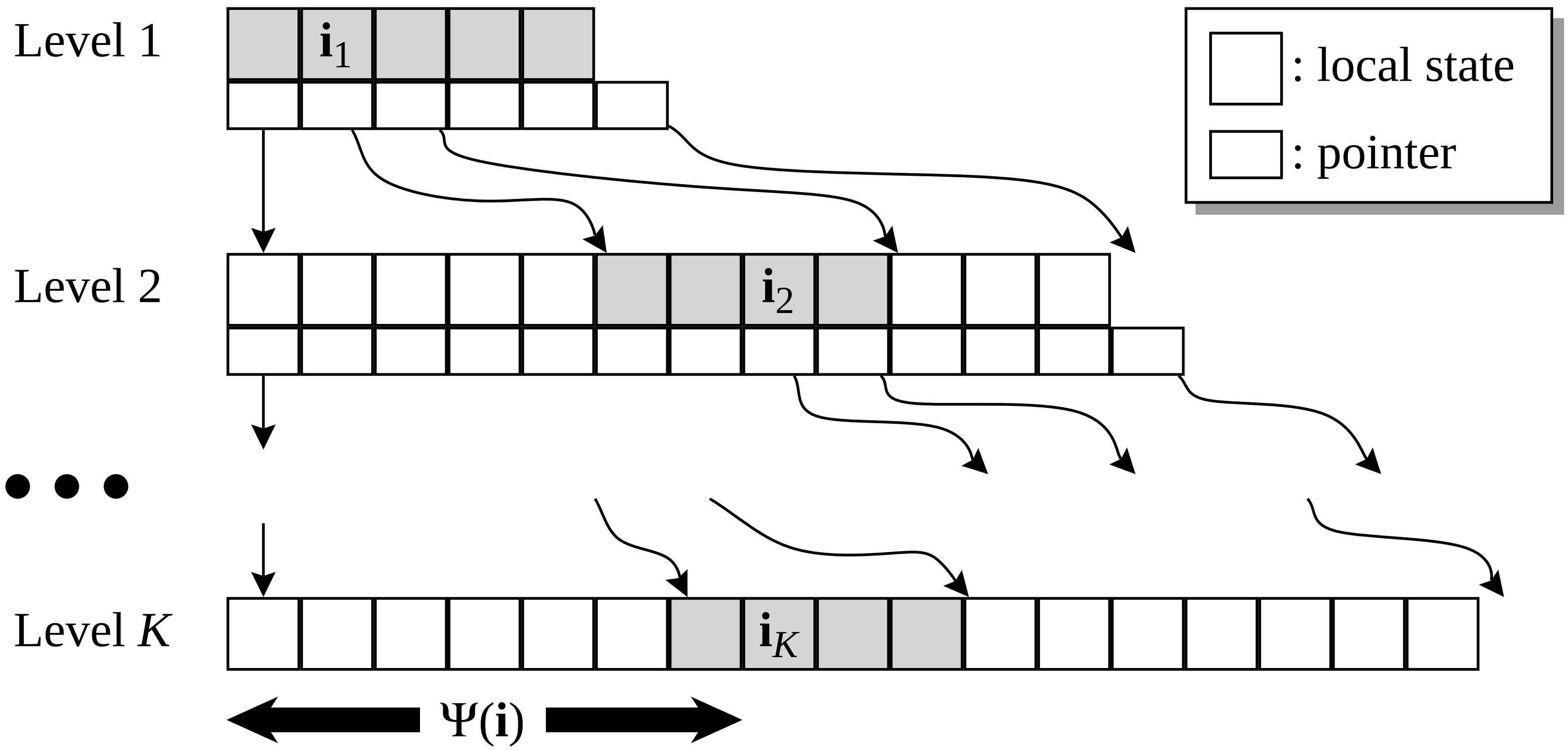




memory requirements: little over 3, 6, or 12 bytes per state

Compressing $\mathcal{S}$ after generation

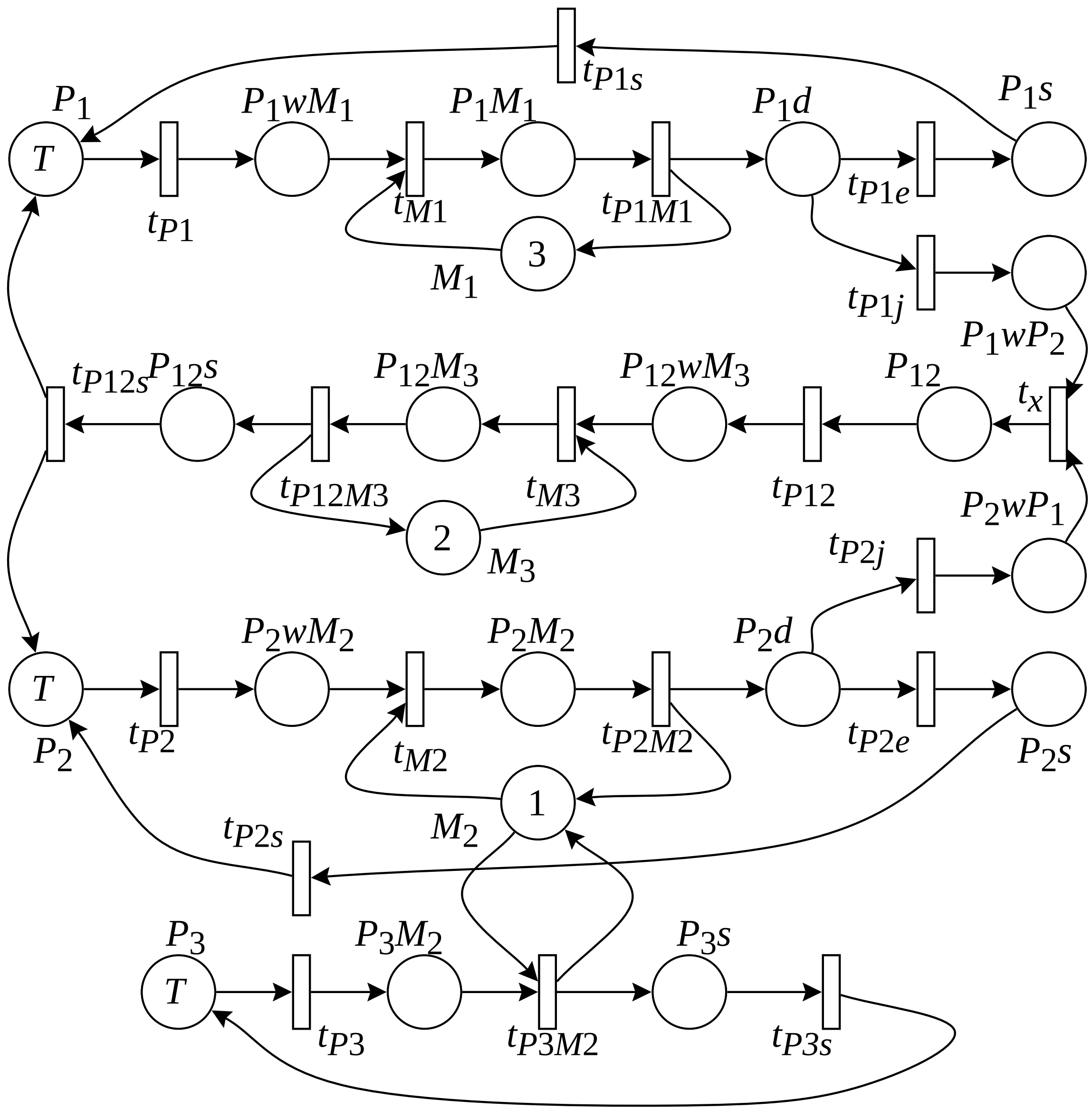
Once $\mathcal{S}$ has been built, we can compress it using arrays:



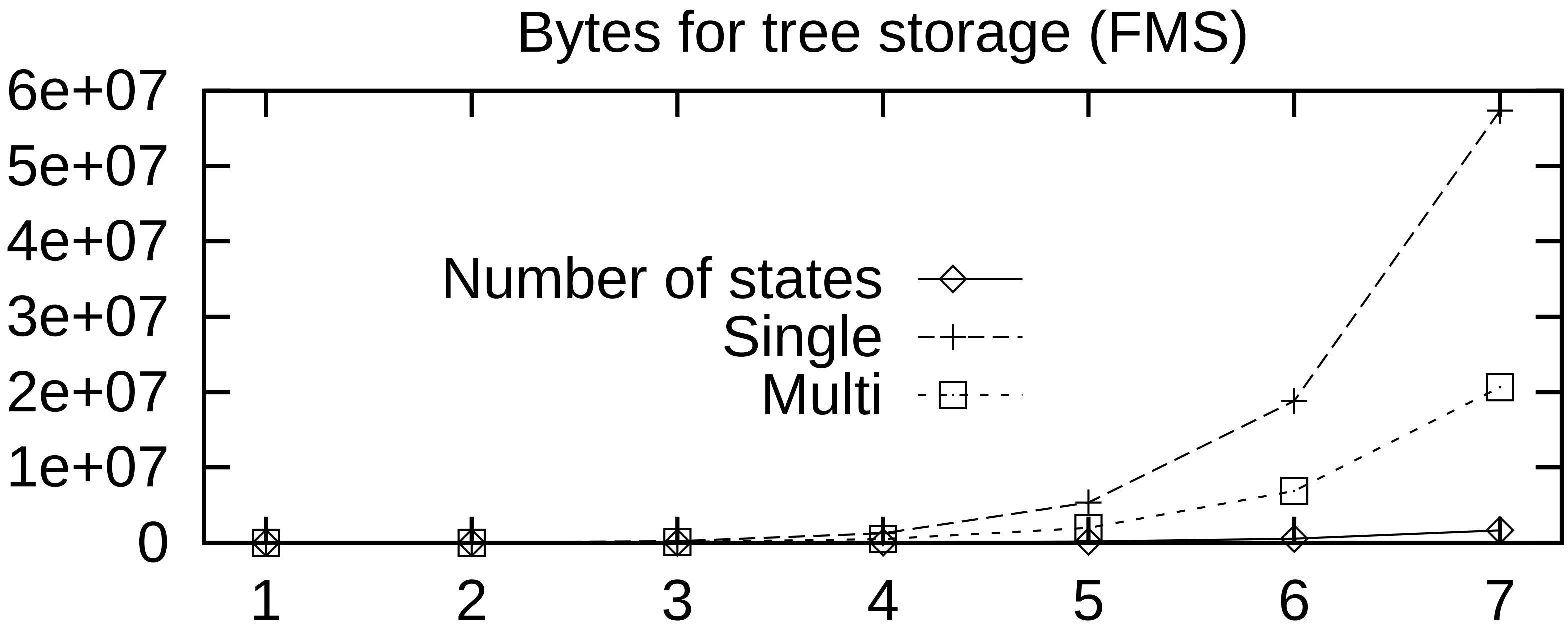
the distance $\psi(\mathbf{i})$ is the lexicographic index of $\mathbf{i}$ in $\mathcal{S}$

memory requirements: little over 1, 2, or 4 bytes per state

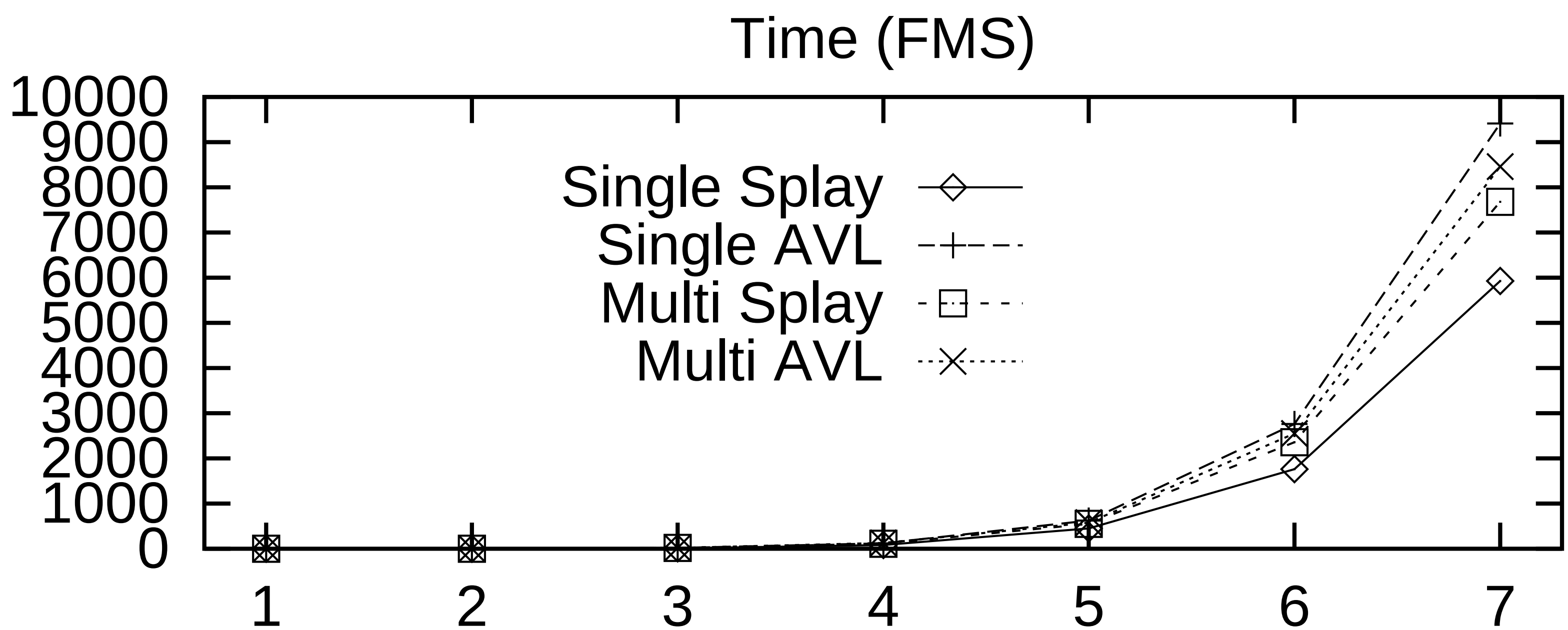
Example for results: a flexible manufacturing system



[redacted]

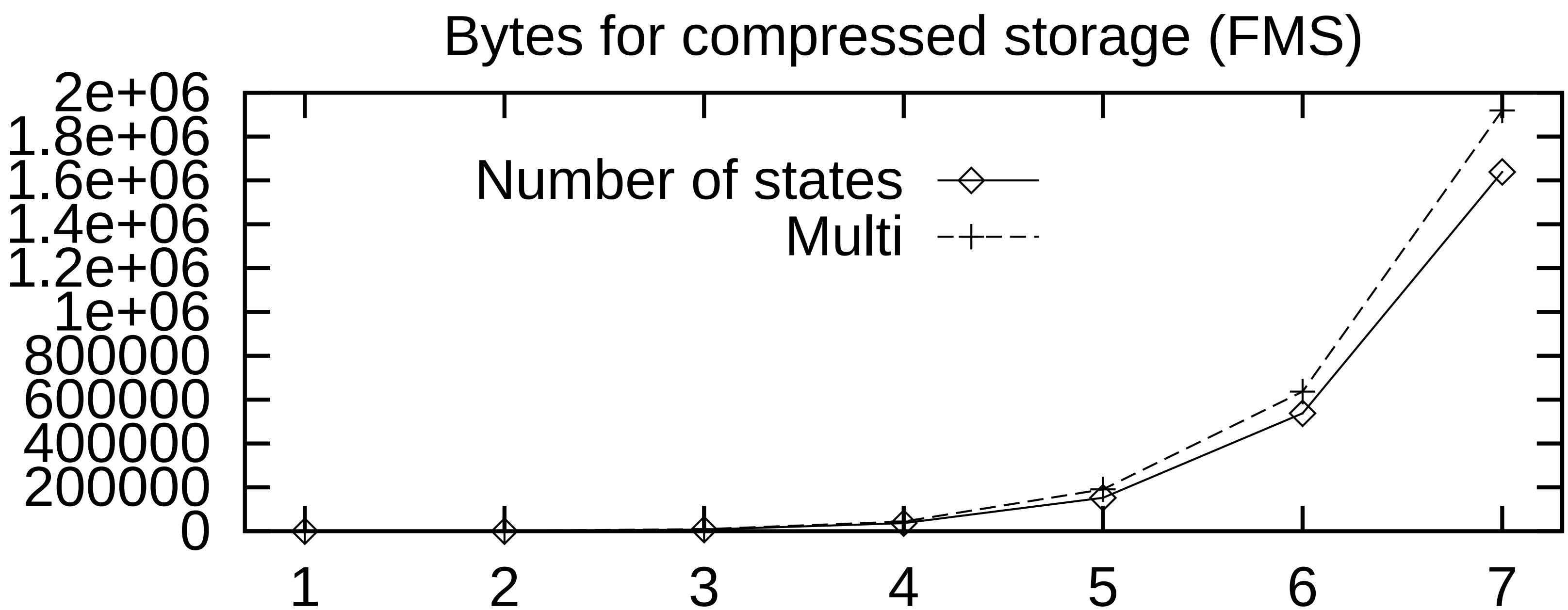


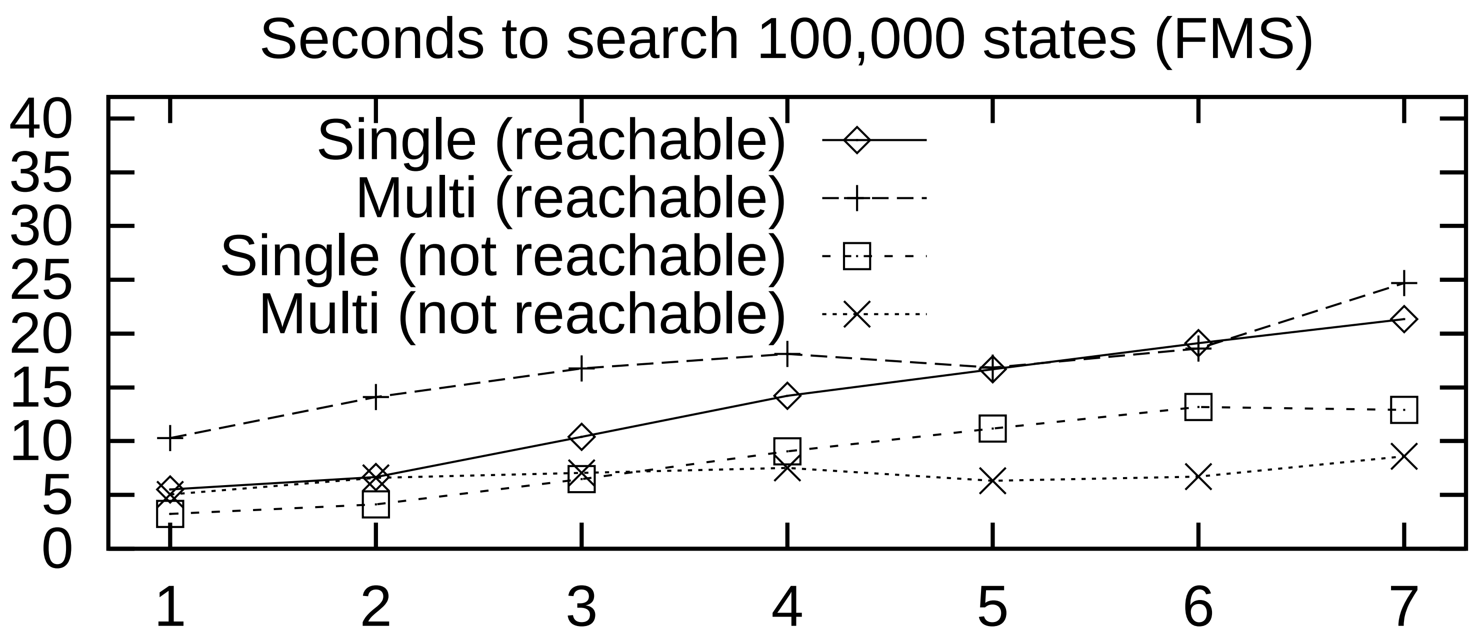
Bytes for tree storage (FMS)
6e+07
5e+07
4e+07
3e+07
2e+07
1e+07
0
1
2
3
4
5
6
7
Number of states
Single
Multi



Time (FMS)
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
1
2
3
4
5
6
7
Single Splay
Single AVL
Multi Splay
Multi AVL

Results for compression and state search



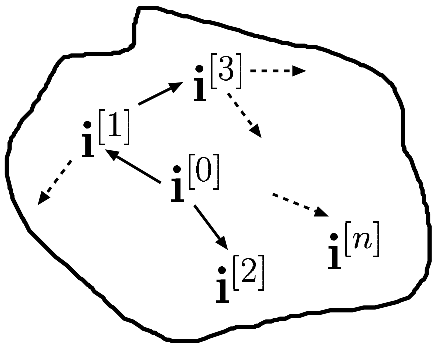


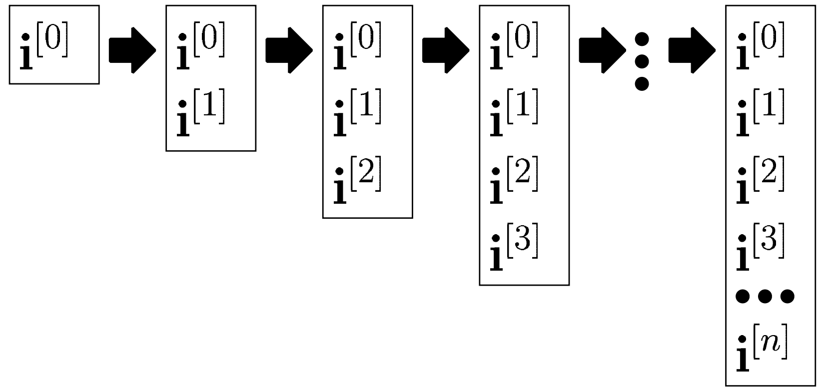
CAN WE DO BETTER THAN THIS?

State-by-state vs. decision-diagram-based generation

Explicit generation of $\mathcal{S}$ adds *one state* at a time

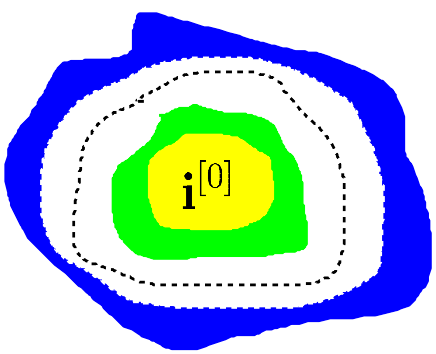
memory increases monotonically

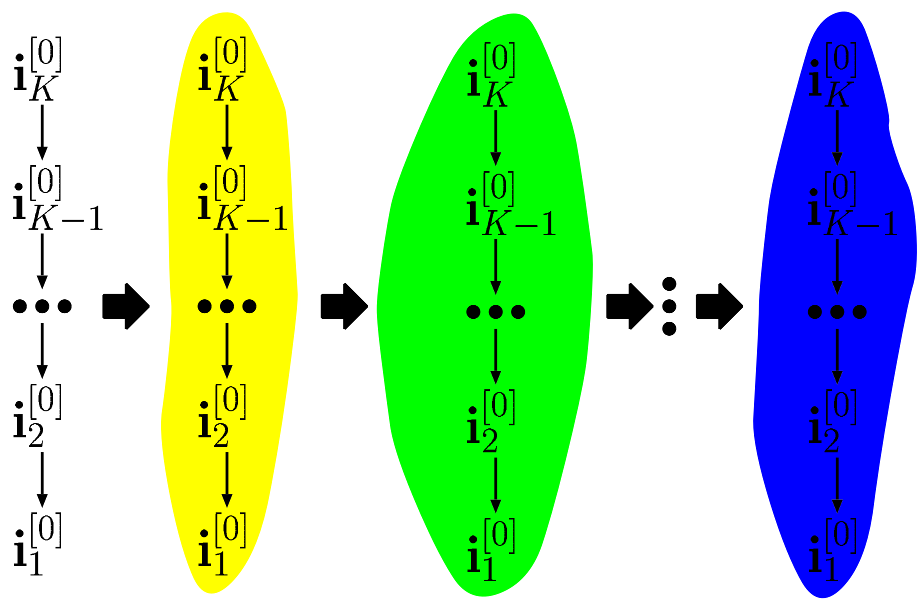




With decision diagrams, we add *sets of states* at each step

memory expands and shrinks





Decision-diagram-based generation of $\mathcal{S}$

$ExploreSymbolic(\mathbf{s}^{init}, \mathcal{N})$ is

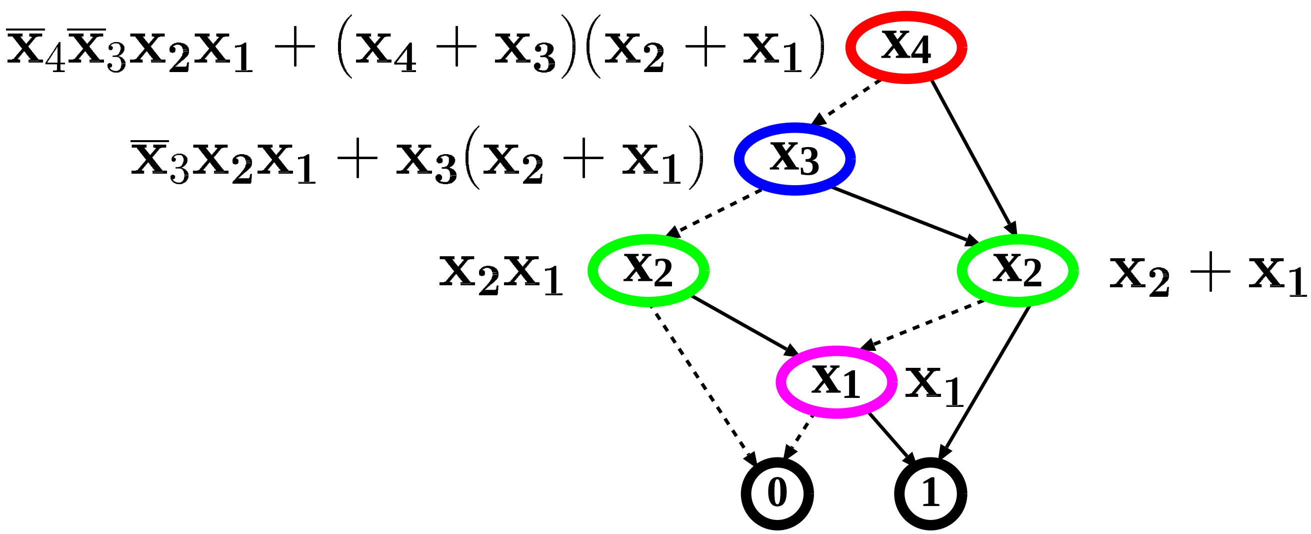
1. $\mathcal{S} \leftarrow \{\mathbf{s}^{init}\}$;
2. repeat
3. $\quad \mathcal{O} \leftarrow \mathcal{S}$; • *old set of known states*
4. $\quad \mathcal{S} \leftarrow \mathcal{O} \cup \mathcal{N}(\mathcal{O})$; • *current set of known states*
5. until $\mathcal{O} = \mathcal{S}$;
6. return $\mathcal{S}$;

with decision diagrams, these set operations can be efficient

(Reduced ordered) binary decision diagrams

Definition of *(RO)BDD*, a *canonical* representation of boolean functions:

- 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 and 1
- 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)*



Structural decomposition of a discrete-state model

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

- the next-state function is partitioned into $\mathcal{N} = \bigcup_{e \in \mathcal{E}} \mathcal{N}_e$
- the global state $\mathbf{i}$ is partitioned into K local states $\mathbf{i} = (\mathbf{i}_K, \ldots, \mathbf{i}_1)$
- so that $\mathcal{S} \subseteq \widehat{\mathcal{S}} = \mathcal{S}_K \times \cdots \times \mathcal{S}_1$
- and, more importantly, $\mathcal{N}_e(\mathbf{i}) = \mathcal{N}_{e,K}(\mathbf{i}_K) \times \cdots \times \mathcal{N}_{e,1}(\mathbf{i}_1)$

a very mild requirement in practice:

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

(Quasi-reduced ordered) multi-valued decision diagrams

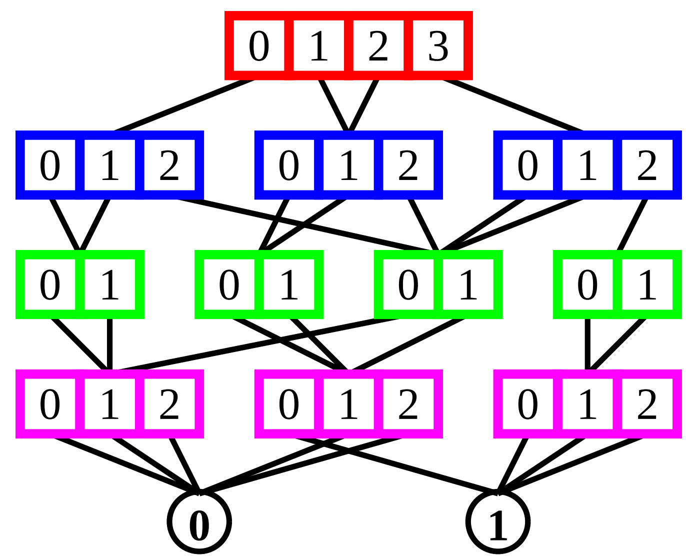
- Nodes are organized into $K+1$ levels
  - Level K contains only one root node
  - Levels $K-1$ through 1 contain one or more nodes
  - Level 0 contains the only two terminal nodes, $\mathbf{0}$ and $\mathbf{1}$ (false and true).
- For $k>0$, a node at level k has $|\mathcal{S}_k|$ arcs pointing to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

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

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

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

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



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

The $Union$ operator for MDDs

If the MDDs a and b encode the sets $\mathcal{A}$ and $\mathcal{B}$, $Union(a, b)$ returns the MDD encoding $\mathcal{A} \cup \mathcal{B}$

$mdd\ Union(lvl\ k, mdd\ a, mdd\ b)$ is

1. if $k = 0$ then return $a \vee b$; • *a and b are* 0 *or* 1
2. if $a = b$ then return a;
3. if $Cache$ contains entry $\langle (k, a, b) = u \rangle$ then return u;
4. for $i = 0$ to $n_k - 1$ do
5. $\quad q_i \leftarrow Union(k-1, a[i], b[i])$;
6. end for
7. $u \leftarrow UniqueTableInsert(k, q_0, \ldots, q_{n_k-1})$;
8. enter $\langle (k, a, b) = u \rangle$ in $Cache$;
9. return u;

Unique Table: determines whether a node we just created is a duplicate

Operation Cache: achieves efficiency. If we did not look it up we would potentially **travel every path** instead of **visit every node** in the MDD

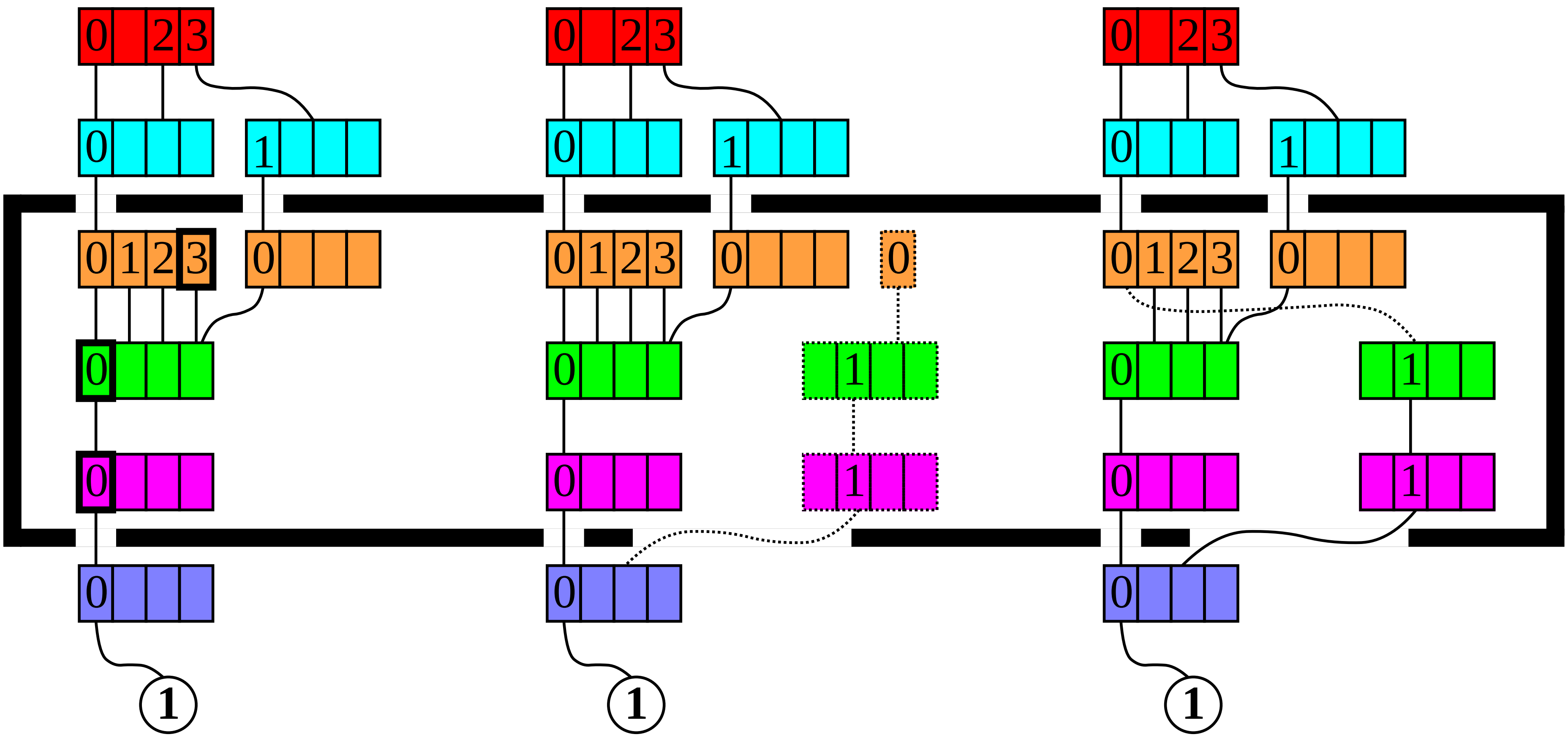
The function $Intersection(a, b)$ differs from $Union(a, b)$ only in the terminal case:

$Union$:
if $k = 0$ then return $a \vee b$;

$Intersection$:
if $k = 0$ then return $a \wedge b$;

worst case complexity: $\#nodes(a) \times \#nodes(b)$

Details of event firing



$$\mathcal{S}: \quad \begin{array}{c} [0,0,*,0,0,*] \\ [2,0,*,0,0,*] \\ [3,1,0,0,0,*] \end{array} \qquad \begin{array}{c} [-,-,3,0,0,-] \\ \xrightarrow{e} \\ [-,-,0,1,1,-] \end{array} \qquad \mathcal{S}: \quad \begin{array}{c} [0,0,*,0,0,*] \\ {\color{red}[0,0,0,1,1,*]} \\ [2,0,*,0,0,*] \\ {\color{red}[2,0,0,1,1,*]} \\ [3,1,0,0,0,*] \end{array}$$

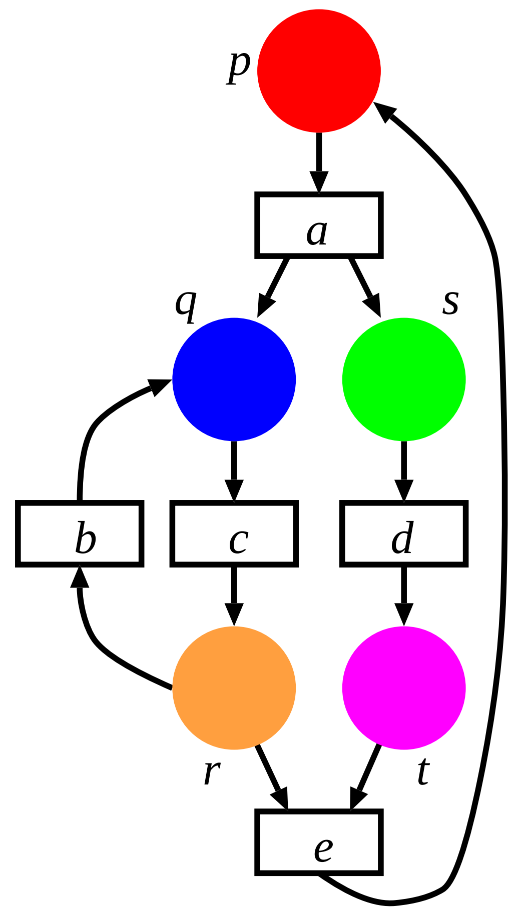
Using structural information to encode $\mathcal{N}$ $\quad (K = 5)$

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

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

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

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



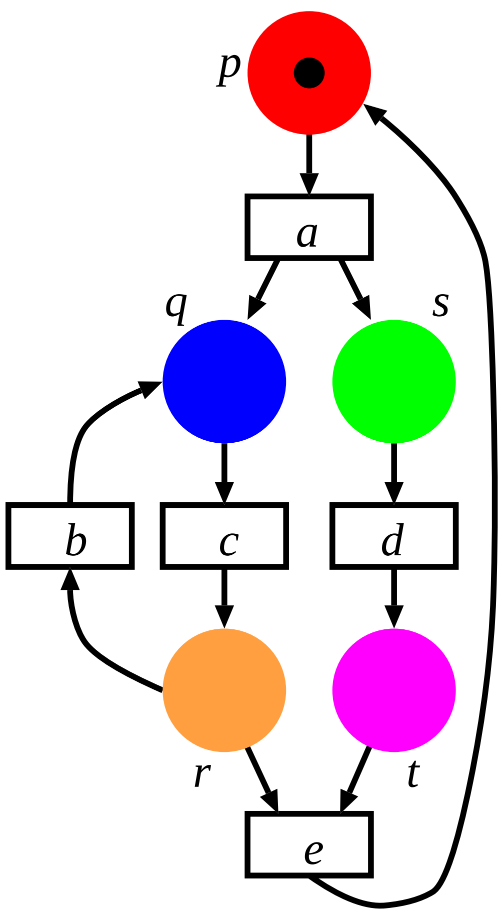
The resulting Kronecker encoding of $\mathcal{N}$ $\quad (K = 5)$

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

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

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

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

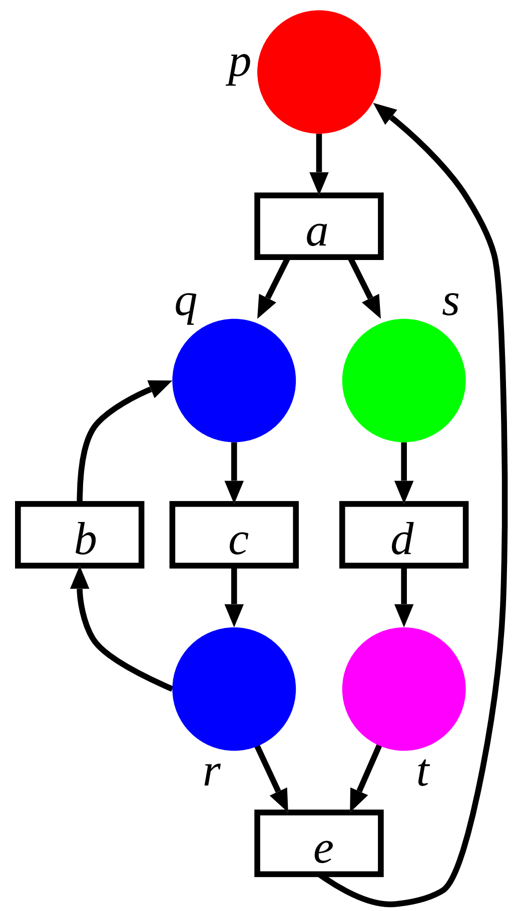


Using structural information to encode $\mathcal{N}$ $(K = 4)$

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

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

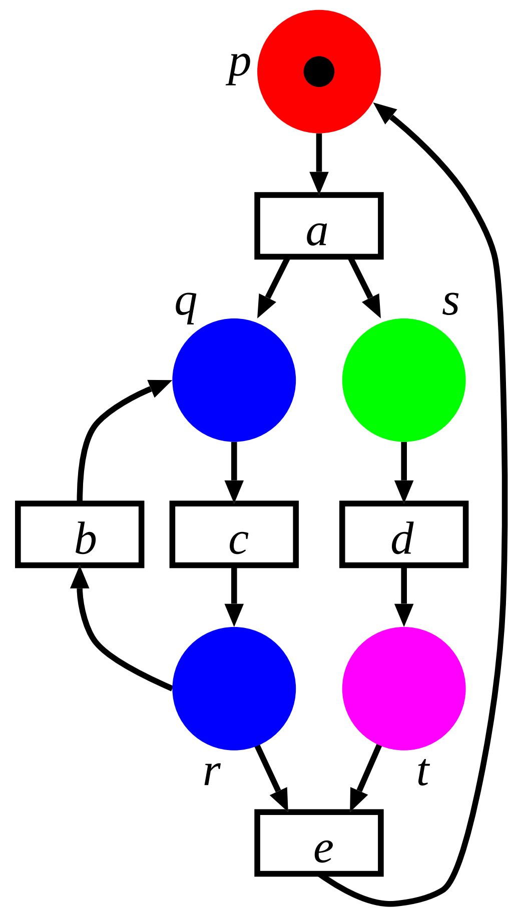
$\mathcal{N}_{a,4}$:?			$\mathcal{N}_{e,4}$:?
$\mathcal{N}_{a,3}$:?	$\mathcal{N}_{l,3}$:?		$\mathcal{N}_{e,3}$:?
$\mathcal{N}_{a,2}$:?		$\mathcal{N}_{d,2}$:?	
		$\mathcal{N}_{d,1}$:?	$\mathcal{N}_{e,1}$:?
$Top(a) : 4$ $Bot(a) : 2$	$Top(l) : 3$ $Bot(l) : 3$	$Top(d) : 2$ $Bot(d) : 1$	$Top(e) : 4$ $Bot(e) : 1$



The resulting Kronecker encoding of $\mathcal{N}$ $\quad (K = 4)$

$\mathcal{S}_4 = \{0{,}1\}$ $\quad$ $\mathcal{S}_3 = \{(0q{,}0r){,}(1q{,}0r){,}(0q{,}1r)\} = \{0{,}1{,}2\}$ $\quad$ $\mathcal{S}_2 = \{0{,}1\}$ $\quad$ $\mathcal{S}_1 = \{0{,}1\}$

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



Definition of Kronecker product

Given K matrices $\mathbf{A}_k \in \mathbb{R}^{n_k \times n_k}$, their *Kronecker product* is

$$\mathbf{A} = \bigotimes_{k=1}^{K} \mathbf{A}_k \in \mathbb{R}^{n_{1:K} \times n_{1:K}}$$

where we define $n_{l:k} = n_l \cdot n_{l+1} \cdots n_k$ and

- $\mathbf{A}[\mathbf{i}, \mathbf{j}] = \mathbf{A}_1[\mathbf{i}_1, \mathbf{j}_1] \cdot \mathbf{A}_2[\mathbf{i}_2, \mathbf{j}_2] \cdots \mathbf{A}_K[\mathbf{i}_K, \mathbf{j}_K]$
- using the mixed-base numbering scheme (indices start at 0)

$$\mathbf{i} = (...((\mathbf{i}_1) \cdot n_2 + \mathbf{i}_2) \cdot n_3 \cdots) \cdot n_K + \mathbf{i}_K = \sum_{k=1}^{K} \mathbf{i}_k \cdot n_{k+1:K}$$

nonzeros: $\eta\left(\bigotimes_{k=1}^{K} \mathbf{A}_k\right) = \prod_{k=1}^{K} \eta(\mathbf{A}_k)$

Kronecker product by example

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

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

$$\left[\begin{array}{ccc|ccc} a_{0,0}b_{0,0} & a_{0,0}b_{0,1} & a_{0,0}b_{0,2} & a_{0,1}b_{0,0} & a_{0,1}b_{0,1} & a_{0,1}b_{0,2} \\ a_{0,0}b_{1,0} & a_{0,0}b_{1,1} & a_{0,0}b_{1,2} & a_{0,1}b_{1,0} & a_{0,1}b_{1,1} & a_{0,1}b_{1,2} \\ a_{0,0}b_{2,0} & a_{0,0}b_{2,1} & a_{0,0}b_{2,2} & a_{0,1}b_{2,0} & a_{0,1}b_{2,1} & a_{0,1}b_{2,2} \\ \hline a_{1,0}b_{0,0} & a_{1,0}b_{0,1} & a_{1,0}b_{0,2} & a_{1,1}b_{0,0} & a_{1,1}b_{0,1} & a_{1,1}b_{0,2} \\ a_{1,0}b_{1,0} & a_{1,0}b_{1,1} & a_{1,0}b_{1,2} & a_{1,1}b_{1,0} & a_{1,1}b_{1,1} & a_{1,1}b_{1,2} \\ a_{1,0}b_{2,0} & a_{1,0}b_{2,1} & a_{1,0}b_{2,2} & a_{1,1}b_{2,0} & a_{1,1}b_{2,1} & a_{1,1}b_{2,2} \end{array}\right]$$

Kronecker product expresses *contemporaneity* or *synchronization*

If $\mathbf{A}$ and $\mathbf{B}$ are the transition probability matrices of two independent discrete-time Markov chains, $\mathbf{A} \otimes \mathbf{B}$ is the transition probability matrix of their composition

Kronecker description of the next-state function

$\mathcal{N}_{e,k} : \mathcal{S}_k \to 2^{\mathcal{S}_k}$ can be identified with a boolean matrix $\mathbf{T}_{e,k} \in \{0,1\}^{|\mathcal{S}_k| \times |\mathcal{S}_k|}$

(a missing $\mathcal{N}_{e,k}$ corresponds to the identity matrix $\mathbf{I}$ of size $|\mathcal{S}_k| \times |\mathcal{S}_k|$)

analogously, $\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}}$ can be identified with a boolean matrix $\widehat{\mathbf{T}} \in \{0,1\}^{|\widehat{\mathcal{S}}| \times |\widehat{\mathcal{S}}|}$

Then,

$$\widehat{\mathbf{T}} = \sum_{e \in \mathcal{E}} \left(\bigotimes_{K \geq k \geq 1} \mathbf{T}_{e,k} \right)$$

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

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

Buchholz, Ciardo, Donatelli, Kemper (INFORMS J. Comp., 2000)

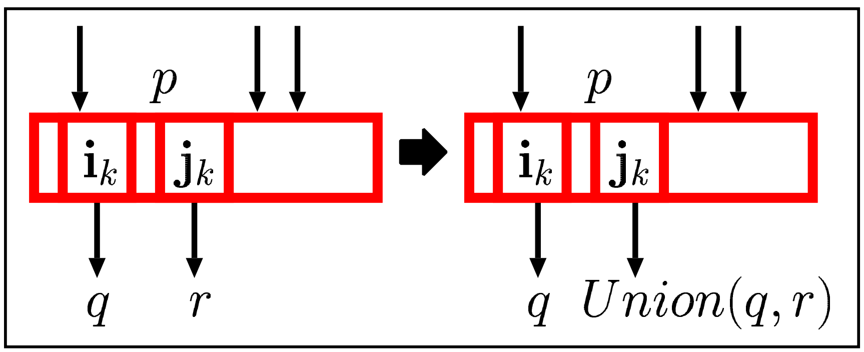
Locality, symmetry, and monotonicity in transition firing

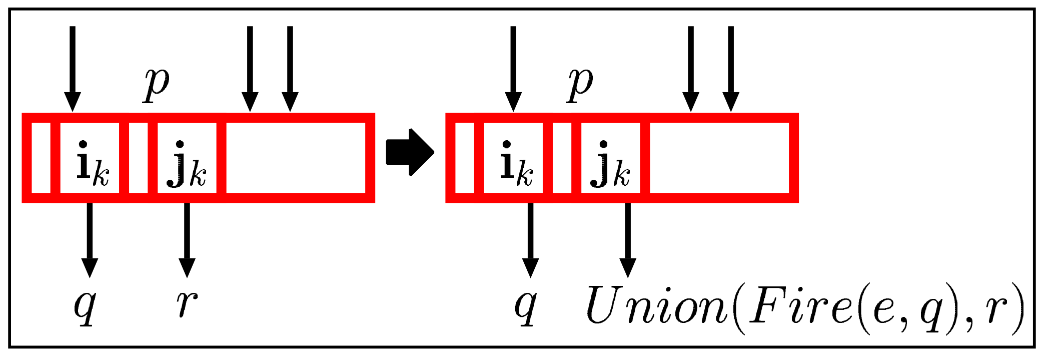
If $\mathbf{i} \in \mathcal{S}$, $\mathbf{i} \xrightarrow{e} \mathbf{j}$, $Top(e)=k \wedge Bot(e)=l$: $\mathbf{j}=(\mathbf{i}_K, \dots, \mathbf{i}_{k+1}, \mathbf{j}_k, \dots, \mathbf{j}_l, \mathbf{i}_{l-1}, \dots, \mathbf{i}_1)$

If also $\mathbf{i}' \in \mathcal{S}$ and $(\mathbf{i}_k, \dots, \mathbf{i}_1)=(\mathbf{i}'_k, \dots, \mathbf{i}'_1)$: $\mathbf{i}' \xrightarrow{e} \mathbf{j}' \wedge \mathbf{j}'=(\mathbf{i}'_K, \dots, \mathbf{i}'_{k+1}, \mathbf{j}_k, \dots, \mathbf{j}_l, \mathbf{i}_{l-1}, \dots, \mathbf{i}_1)$

Local event $\mathbf{i}_k \xrightarrow{e} \mathbf{j}_k$

Synchronizing event $(\mathbf{i}_k, \dots, \mathbf{i}_l) \xrightarrow{e} (\mathbf{j}_k, \dots, \mathbf{j}_k)$





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

Saturation: an efficient iteration strategy

Traditional application of a partitioned $\mathcal{N}$:

$$\begin{array}{rcl} \mathcal{X}^{(e_1)} & \leftarrow & \mathcal{N}_{e_1}(\mathcal{S}) \\ & \cdots & \\ \mathcal{X}^{(e_{|\mathcal{E}|})} & \leftarrow & \mathcal{N}_{e_{|\mathcal{E}|}}(\mathcal{S}) \\ \mathcal{S} & \leftarrow & \mathcal{S} \cup \mathcal{X}^{(e_1)} \cup \ldots \cup \mathcal{X}^{(e_{|\mathcal{E}|})} \end{array}$$

We can improve by *pipelining*:

$$\begin{array}{rcl} \mathcal{S} & \leftarrow & \mathcal{S} \cup \mathcal{N}_{e_1}(\mathcal{S}) \\ & \cdots & \\ \mathcal{S} & \leftarrow & \mathcal{S} \cup \mathcal{N}_{e_{|\mathcal{E}|}}(\mathcal{S}) \end{array}$$

And even more by *exhaustive pipelining*:

$$\begin{array}{rcl} \mathcal{S} & \leftarrow & \mathcal{S} \cup \mathcal{N}^*_{e_1}(\mathcal{S}) \\ & \cdots & \\ \mathcal{S} & \leftarrow & \mathcal{S} \cup \mathcal{N}^*_{e_{|\mathcal{E}|}}(\mathcal{S}) \end{array}$$

But the best strategy is to *saturate* MDD nodes recursively bottom-up:

- a node at level k is saturated if it is a fixed point w.r.t. all events e s.t. $Top(e) \leq k$
- traditional idea of a global fixed-point iteration for the overall MDD disappears

enormous savings in both time and (peak) memory

Merging explicit local with symbolic global s.s. generation

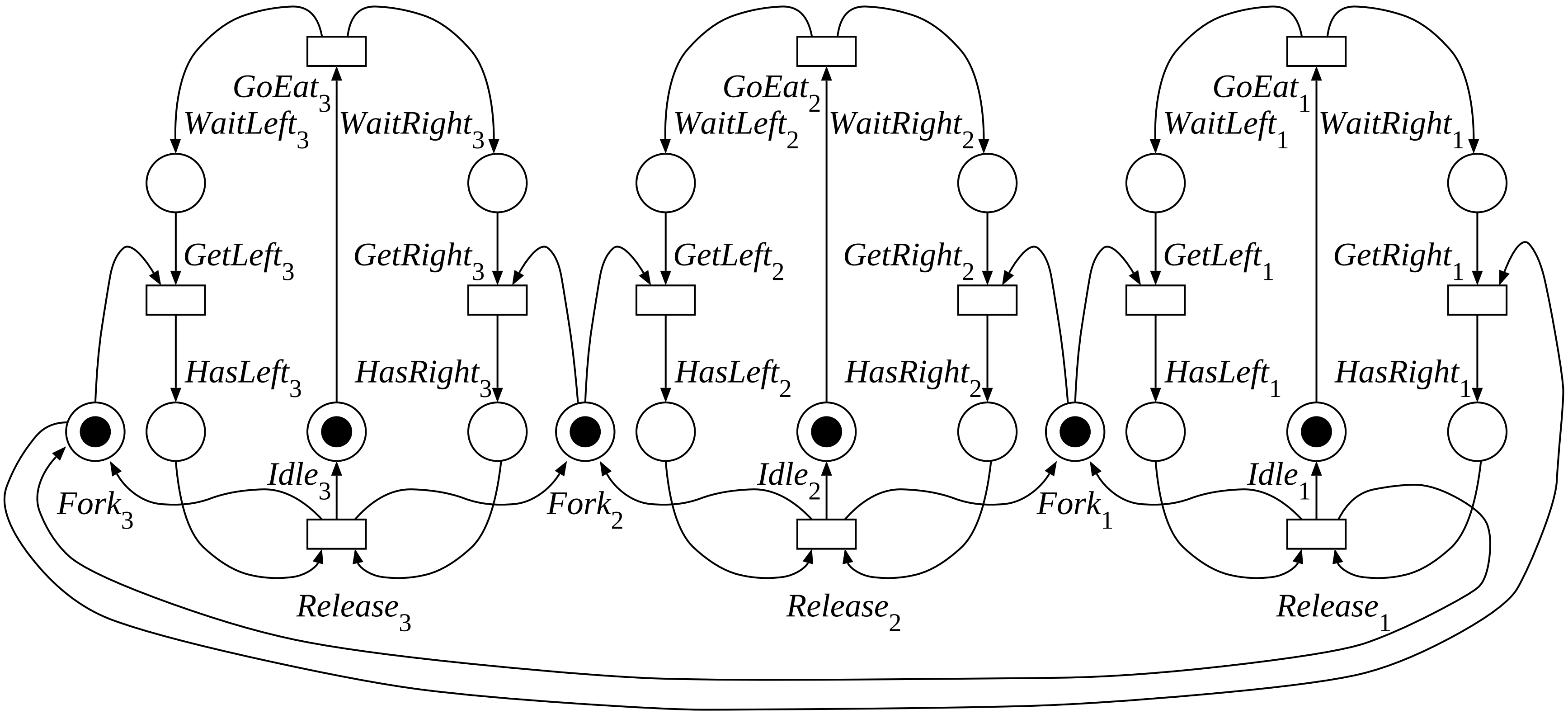
Problem: local state spaces $\mathcal{S}_k$ are not known a priori

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

1. start from the only known state, the initial state $(\mathbf{s}_K, \ldots, \mathbf{s}_1)$, and *commit* its components
2. while MDD encoding $\mathcal{S}$ has not reached its fixed point w.r.t. $\mathcal{N}$ do
3. (explicitly) explore all $\mathbf{j}_k$ reachable from each newly committed $\mathbf{i}_k$ in isolation *in one step*
   $\Rightarrow$ *create corresponding row* $\mathbf{i}_k$ *of* $\mathcal{N}_{e,k}$ *for each* $e \in \mathcal{E}$ *dependent on level* k
4. (symbolically) explore global states reachable from the currently-known $\mathcal{S}$
   $\Rightarrow$ *use current* $\mathcal{N}_{e,k}$ *matrices*
   $\Rightarrow$ *may cause uncommitted local states to be committed*
5. end while

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

Example: the dining philosophers (Petri net)



N subnets connected in a circular fashion

Example: the dining philosophers (SMART code)


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

Example: the dining philosophers (results)


```
Reading input.
N=50
Reachable states: 22,291,846,172,619,859,445,381,409,012,498
Good states:      22,291,846,172,619,859,445,381,409,012,496
The bad states are

State 0 : { WaitR[0]:1 HasL[0]:1 WaitR[1]:1 HasL[1]:1 WaitR[2]:1 HasL[2]:1 Wa
State 1 : { WaitL[0]:1 HasR[0]:1 WaitL[1]:1 HasR[1]:1 WaitL[2]:1 HasR[2]:1 Wa
Done.
```

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

N	Reachable states	Final memory (kB)		Peak memory (kB)		Time (sec)	
		SMART	NuSMV	SMART	NuSMV	SMART	NuSMV
Dining Philosophers (*N* levels)							
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 (*N* levels)							
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 (*N*+1 levels)							
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 (19 levels)							
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	—

Symbolic model checking

We can talk about events and states occurring over *relative time*, or *temporal logic*

- Can event e ever fire before event f?
- Is it possible to reach a state where both buffers are empty?
- Once both buffers are empty, can they ever both become full at the same time?
- Or even just at different times?
- Can we reach a stable set of states where race conditions cannot occur?
- Can we reach a set of states where, if race conditions occur, they never cause a deadlock?

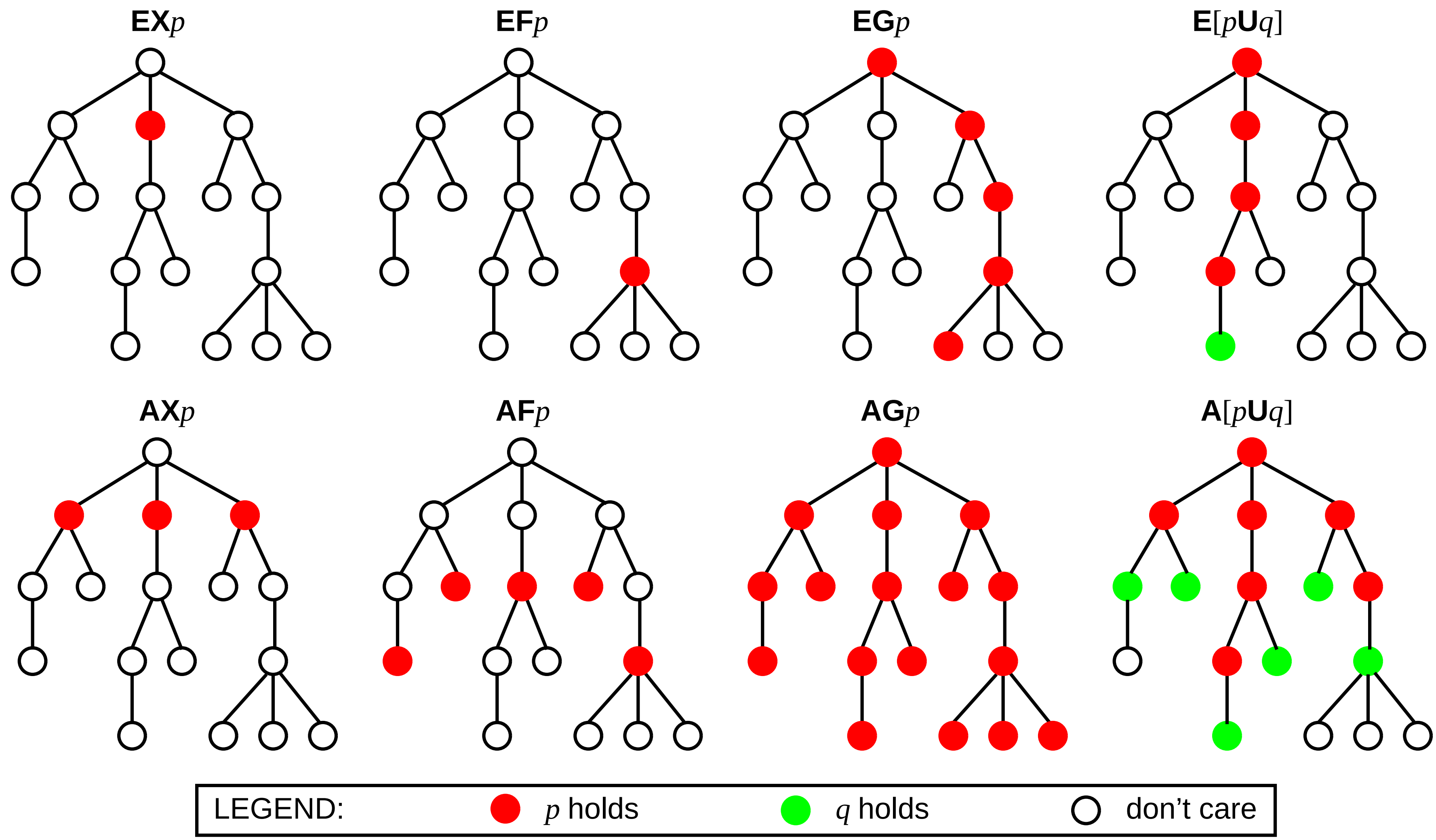
We use *computation tree logic (CTL)* to express these queries:

- Any atomic proposition (true or false in a state) is a CTL formula
- If p and q are CTL formulas, so are $\neg p$, $p \wedge q$, $p \vee q$
- If p and q are CTL formulas, so are EXp, EFp, EGp, E$[p$U$q]$, AXp, AFp, AGp, A$[p$U$q]$

given a model, a CTL formula p identifies a set of states

(those states that satisfy p)

CTL semantics



Note that EX, EG, and EU is a complete set of CTL operators, since

$$\text{EF}p = \text{E}[\top \text{U} p] \qquad \text{AX}p = \neg\text{EX}\neg p \qquad \text{AF}p = \neg\text{EG}\neg p$$

$$\text{AG}p = \neg\text{E}[\top \text{U} \neg p] \qquad \text{A}[p\text{U}q] = \neg\text{E}[\neg q \text{U} \neg p \wedge \neg q] \wedge \neg\text{EG}\neg q$$

Applications

Protocol verification

Security

Software correctness

VLSI design and verification

GUI and HCI testing