Combining structural and symbolic methods for the verification of concurrent systems

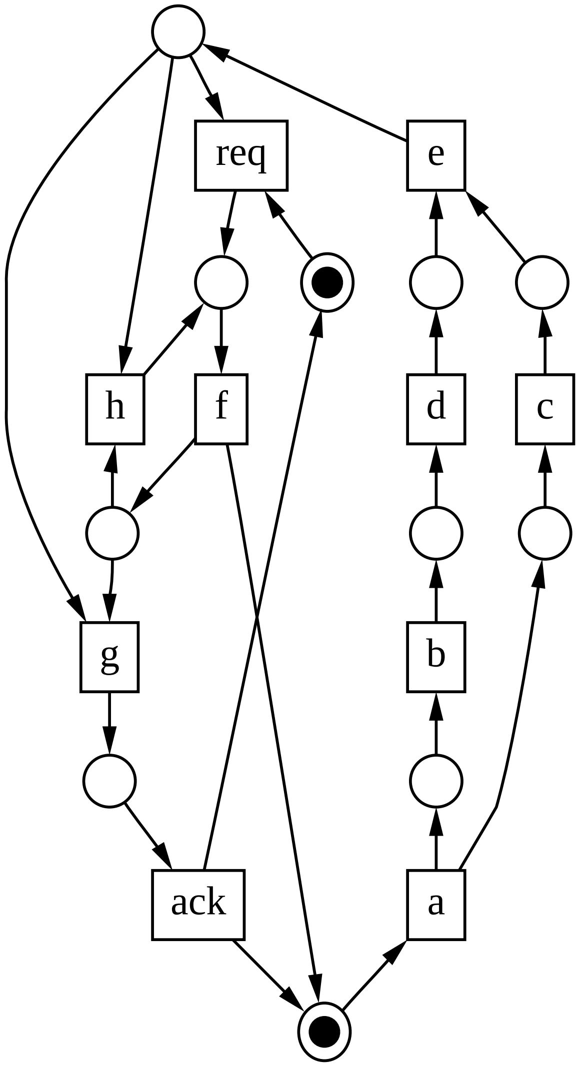
Jordi Cortadella

Department of Software
University Politècnica de Catalunya
Barcelona, Spain

Thanks to:

- Michael Kishinevsky (The University of Aizu, Japan)
- Alex Kondratyev (The University of Aizu, Japan)
- Luciano Lavagno (Politecnico di Torino, Italy)
- Enric Pastor (Universitat Politècnica de Catalunya, Spain)
- Oriol Roig (Universitat Politècnica de Catalunya, Spain)
- Alex Taubin (The University of Aizu, Japan)
- Alex Yakovlev (University of Newcastle upon Tyne, UK)

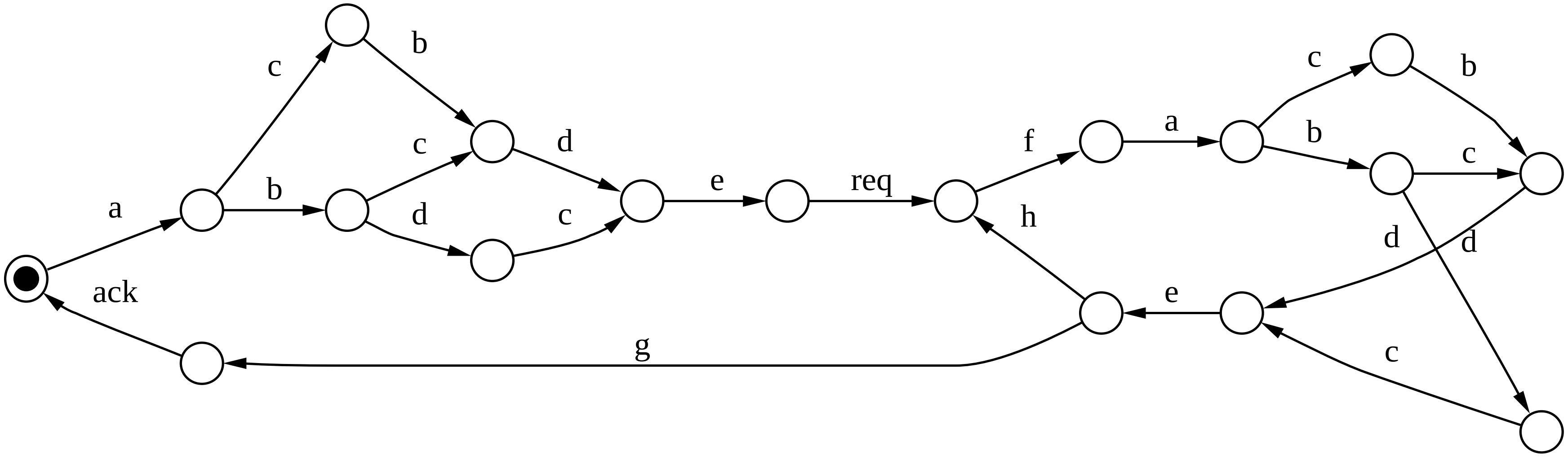
Motivation: formal verification



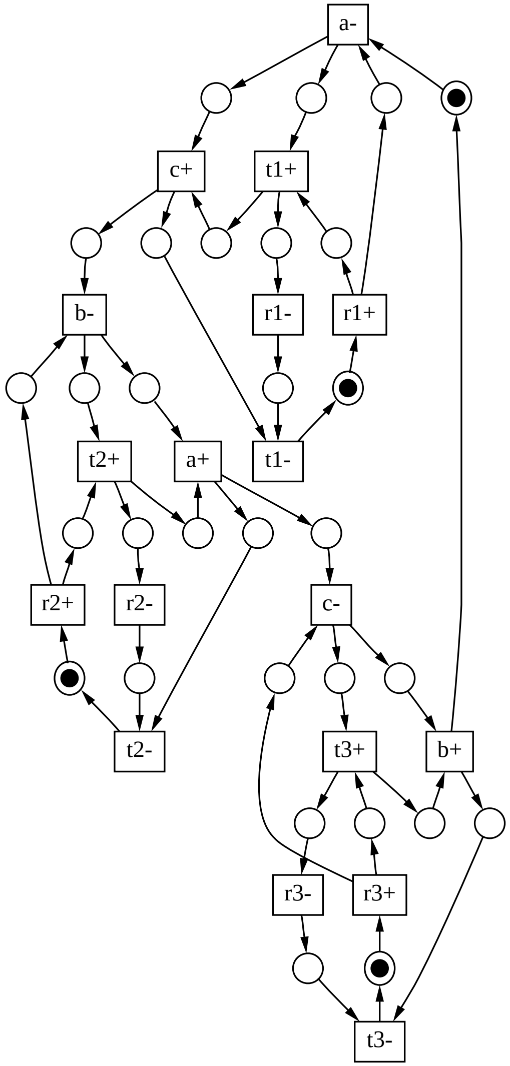
Is every *request* eventually *acknowledged* ?

$$\mathbf{AG}(req \Rightarrow \mathbf{AF}ack) \quad ?$$

Motivation: formal verification



Motivation: synthesis

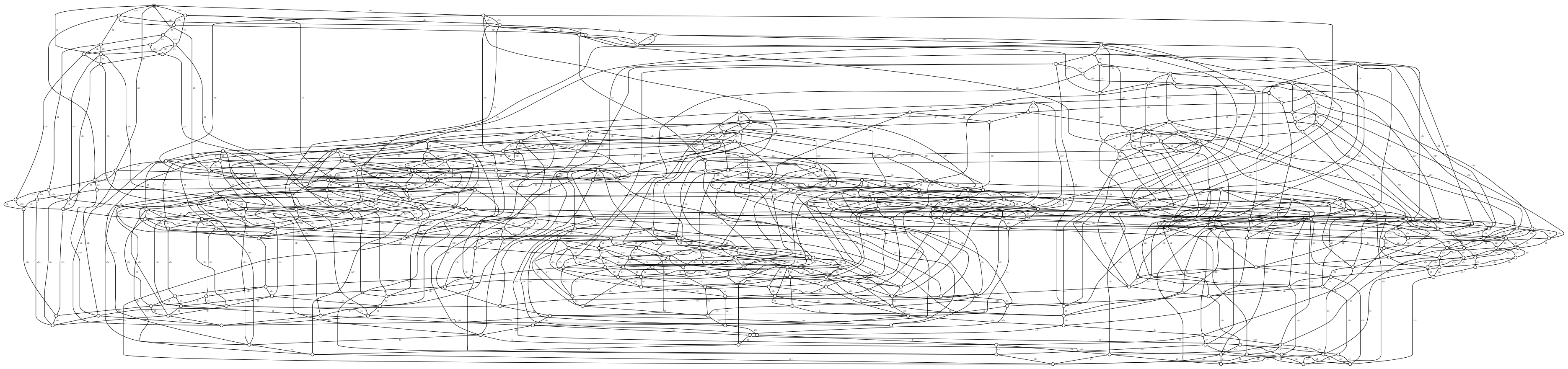


Logic equations:

$$
\begin{aligned}
a &= a(\overline{r_1} + t_1 + \overline{b}) + \overline{b}t_2 \\
b &= b(\overline{r_2} + t_2 + \overline{c}) + \overline{c}t_3 \\
c &= c(\overline{a} + t_3 + \overline{r_3}) + \overline{a}t_1 \\
t_1 &= \overline{ac} + t_1\overline{r_1} \\
t_2 &= \overline{a}\overline{b} + t_2\overline{r_2} \\
t_3 &= \overline{b}\overline{c} + t_3\overline{r_3}
\end{aligned}
$$

1. Assignment of binary codes to states
2. Derivation of boolean functions

Motivation: synthesis



This is a bus specification with 9 signals and 336 states.

There are specifications with more than 30 signals and 10^{13} states.

Outline

- Event vs. state-based models
- Symbolic analysis (BDDs)
- Structural theory
- Structural + symbolic analysis
  - State encoding
  - Reachability analysis
- Some experimental results
- A synthesis/verification framework
- Conclusions

Our problem

- We need to manipulate information at state level.
- The state space grows exponentially with concurrency.
- Symbolic methods (BDDs) are not the remedy for all our ills.
- The use of BDDs (brute force) sometimes manifests the lack of clever strategies to represent and manipulate the state space.
- Unfortunately, we often need to manipulate the **exact** (or almost exact) state space.

Event vs. State-based models (I)

- **Event-based models:**
  - Concurrency is explicit: $e; (a \parallel b \parallel c)$
  - State is implicit
  - Examples: CSP, Petri nets, CCS
- **State-based models:**
  - State is explicit: $eabc + eacb + ebac + ebca + ecab + ecba$
  - Concurrency is implicit (concurrency = interleaving)
  - Examples: FSMs

Event vs. State-based models (II)

	Event-based	State-based
good	• Succint representation • Causal/concurrent relations are explicit • Easy specification	• Detailed information at state level • Easy for synthesis, analysis and verification
bad	• No detailed information at state level • difficult analysis, synthesis and verification	• State explosion • Causal/concurrent relations are not explicit

Event and state-based models are interchangeable:

Event-based (Petri nets) $\Longleftrightarrow$ State-based (transition systems)

. . . but at exponential cost

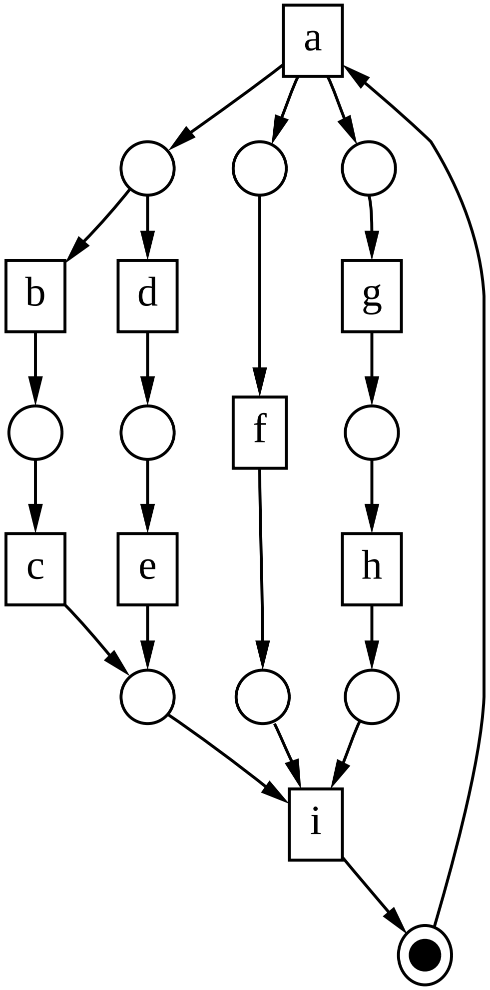
Event vs. State-based models (III)

But we need both models. Examples:

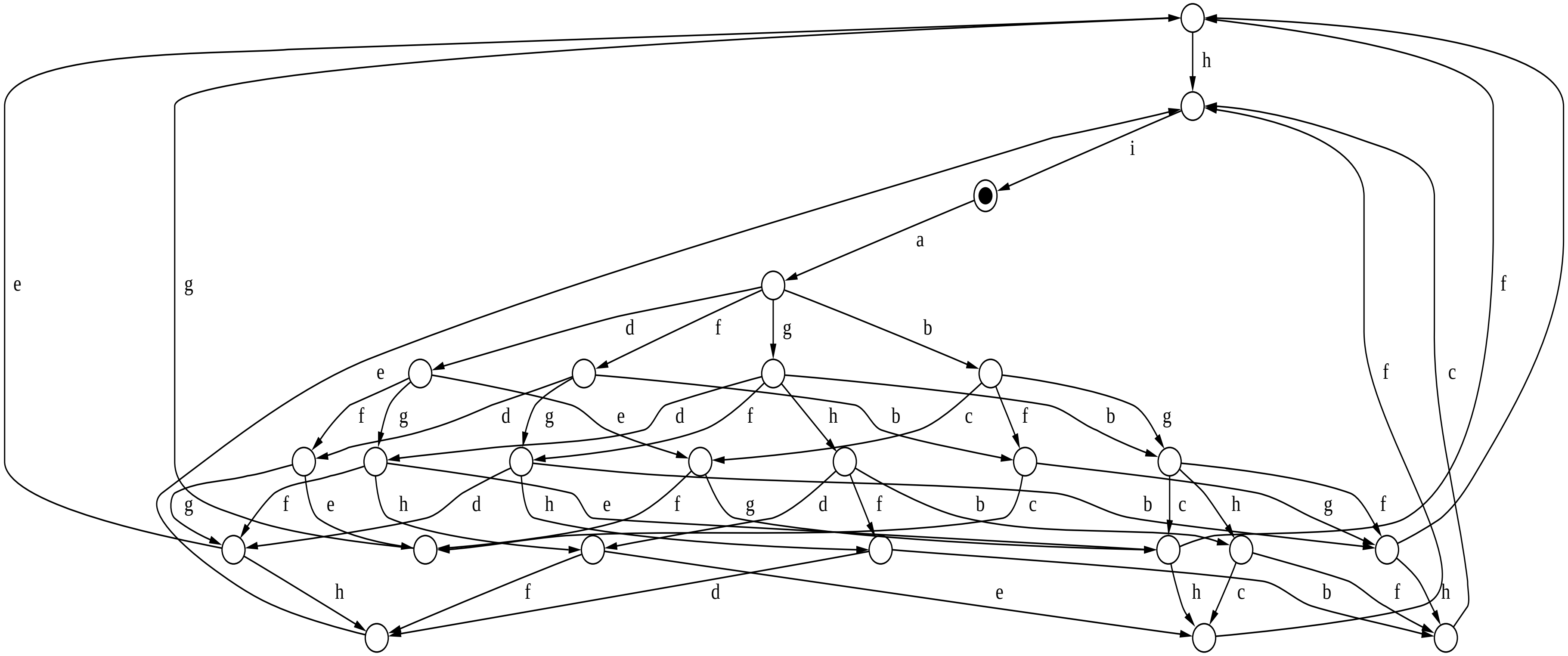
- **Event-based:**
  - Specification of asynchronous circuits (STGs), distributed algorithms (CSP), etc ...
  - Graphical representation (Petri nets).

- **State-based:**
  - Synthesis of asynchronous circuits (binary encoding of states)
  - Verification of temporal logic formulae by fixpoint methods: $\mathbf{AG}(req \Longrightarrow \mathbf{AF} ack)$

Event-based (Petri net) specification



State-based (transition system) specification



Boolean algebra

Boolean algebras

$(\{0,1\}, +, \cdot, 0, 1)$

Switching algebra

$(F_n(B), +, \cdot, 0, 1)$

$F_n(B)$: set of all functions of n boolean variables

Boolean functions

x_1	x_2	x_3	f	
0	0	0	0	
0	0	1	1	$\leftarrow$ minterm
0	1	0	0	
0	1	1	0	
1	0	0	1	
1	0	1	1	
1	1	0	1	
1	1	1	1	

$$f = x_1 + \overline{x}_2 x_3$$

Boolean function $\equiv$ set of minterms

Boolean algebra isomorphism (Stone's theorem)

Fruits = { apple, banana, cherry } (F={a,b,c})

$(2^F, \cup, \cap, \emptyset, F)$

subset:

$$s = \{a, b\}$$

set of subsets:

$$S = \{\{a, b\}, \{c\}, \{a, c\}\}$$

union:

$$S = S_1 \cup S_2$$

intersection:

$$S = S_1 \cap S_2$$

$(F_3(B), +, \cdot, 0, 1)$

minterm:

$$s = ab\overline{c}$$

function:

$$S = ab\overline{c} + \overline{a}\overline{b}c + a\overline{b}c$$

sum:

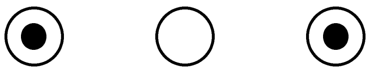
$$S = S_1 + S_2$$

product:

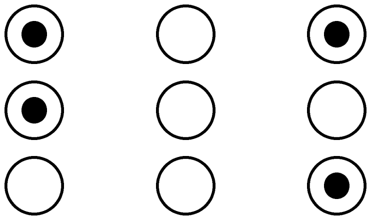
$$S = S_1 \cdot S_2$$

Modeling Petri nets with boolean functions

$p_1 \quad p_2 \quad p_3$



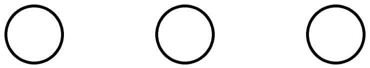
marking $\longrightarrow$ **minterm:** $m = p_1\bar{p}_2p_3$



set of markings $\longrightarrow$

$$\begin{aligned} M_1 &= p_1\bar{p}_2p_3 + \\ & p_1\bar{p}_2\bar{p}_3 + \\ & \bar{p}_1\bar{p}_2p_3 \\ &= \bar{p}_2(p_1 + p_3) \end{aligned}$$

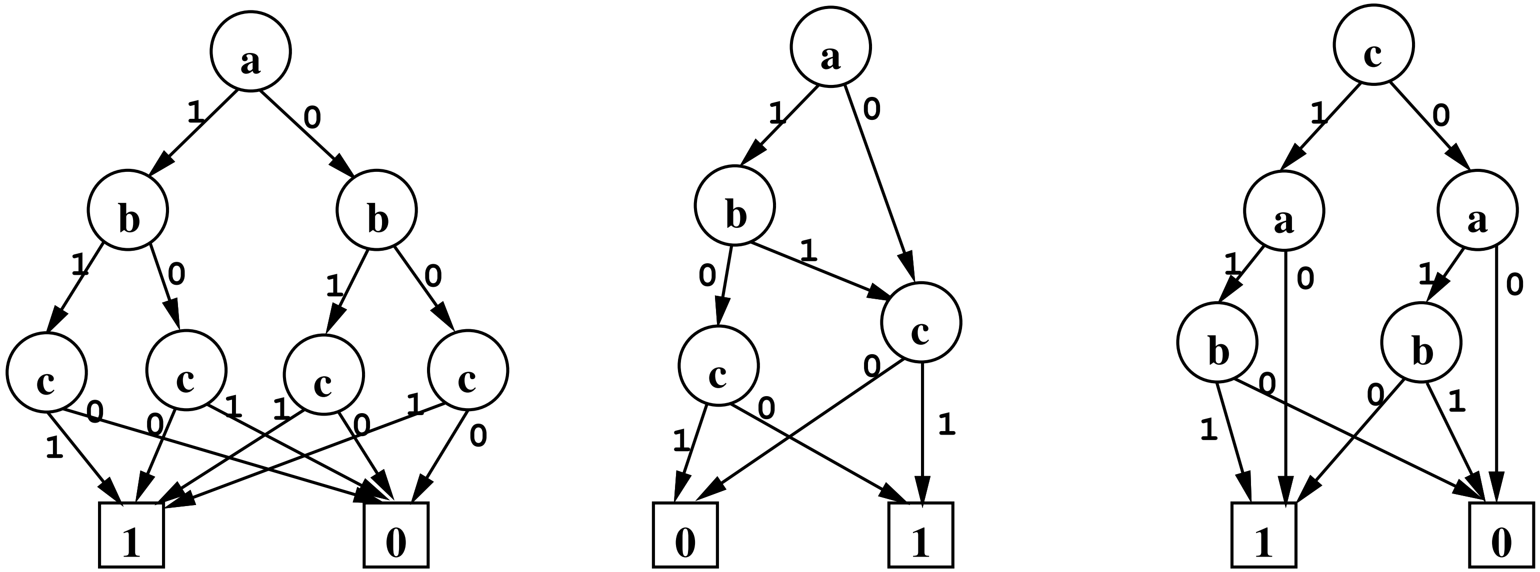
$\cup$



$$M_2 = M_1 + \bar{p}_1\bar{p}_2\bar{p}_3 = \bar{p}_2$$

Binary Decision Diagrams [Bryant 86]

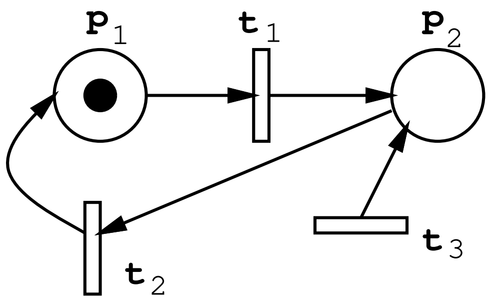
Example: $f = bc + a\bar{b}\bar{c} + \bar{a}c$



- OBDDs: all variables appear in the same order
- Reduced OBDDs (ROBDDs): no isomorphic subgraphs.
- ROBDDs are a canonical form to represent boolean functions.
- ROBDDs are used for efficiently representing logic functions.

Transition firing

Let ${}^\bullet t$ and $t^\bullet$ be the predecessor and successor places of t.



$$
\begin{aligned}
\text{Enabled}_t &= \bigwedge_{p_i \in {}^\bullet t} p_i && (t \text{ enabled}),\\
\text{NPM}_t &= \bigwedge_{p_i \in {}^\bullet t} \bar{p}_i && (\text{no predecessor marked}),\\
\text{ASM}_t &= \bigwedge_{p_i \in t^\bullet} p_i && (\text{all successors marked}),\\
\text{NSM}_t &= \bigwedge_{p_i \in t^\bullet} \bar{p}_i && (\text{no successor marked}).
\end{aligned}
$$

Next state function: $\delta_T : 2^{M_P} \times T \rightarrow 2^{M_P}$

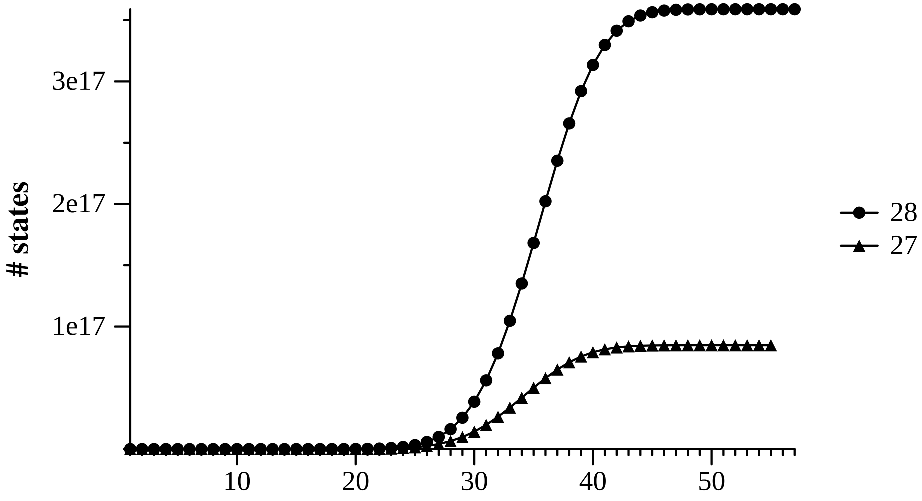
$$
\boxed{\delta_T(M,t) = \left(M_{\text{Enabled}_t} \cdot \text{NPM}_t\right)_{\text{NSM}_t} \cdot \text{ASM}_t}
$$

$$
\begin{aligned}
M &= p_1\bar{p}_2p_3\bar{p}_4\bar{p}_5 + \bar{p}_1p_2p_3\bar{p}_4\bar{p}_5 + p_1\bar{p}_2\bar{p}_3\bar{p}_4p_5\\
M_{\text{Enabled}_{t_1}} \cdot \text{NPM}_{t_1} &= \bar{p}_1\bar{p}_2p_3\bar{p}_4\bar{p}_5 + \bar{p}_1\bar{p}_2\bar{p}_3\bar{p}_4p_5\\
\delta_T(M,t_1) &= \bar{p}_1p_2p_3\bar{p}_4\bar{p}_5 + \bar{p}_1p_2\bar{p}_3\bar{p}_4p_5
\end{aligned}
$$

Petri Net Symbolic Traversal (BFS)

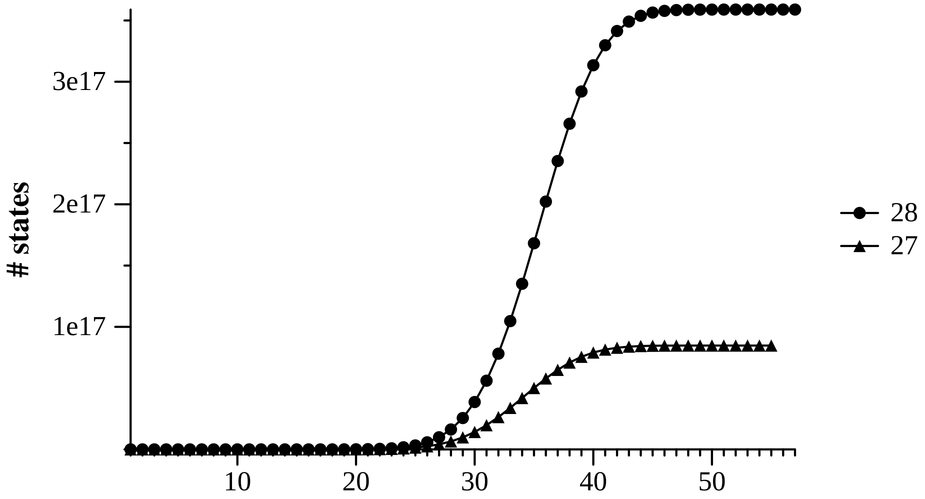
$$\Delta(M) = \bigcup_{\forall t \in T} \delta(M,t) \rightarrow \text{one-step reachability}$$

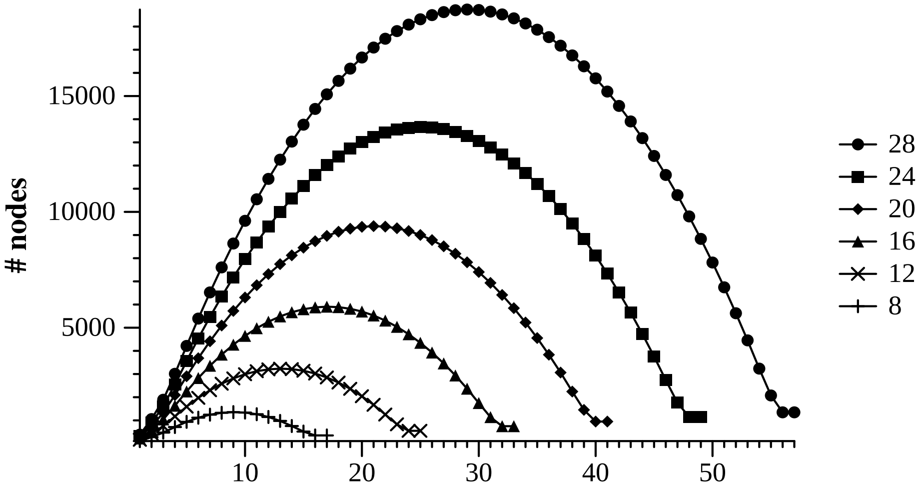
Reached $=$ *From* $= \{m_0\}$;
repeat {
  To $= \Delta($*From*$)$;
  New $=$ *To* $-$ *Reached*;
  From $=$ *New*;
  Reached $=$ *Reached* $\cup$ *New*;
} until (*New* $= \emptyset$);



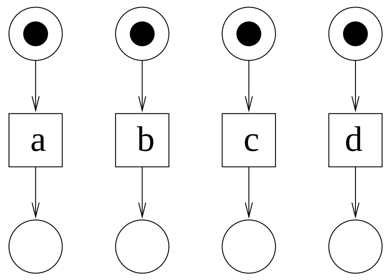
iterations: sequential depth of the net

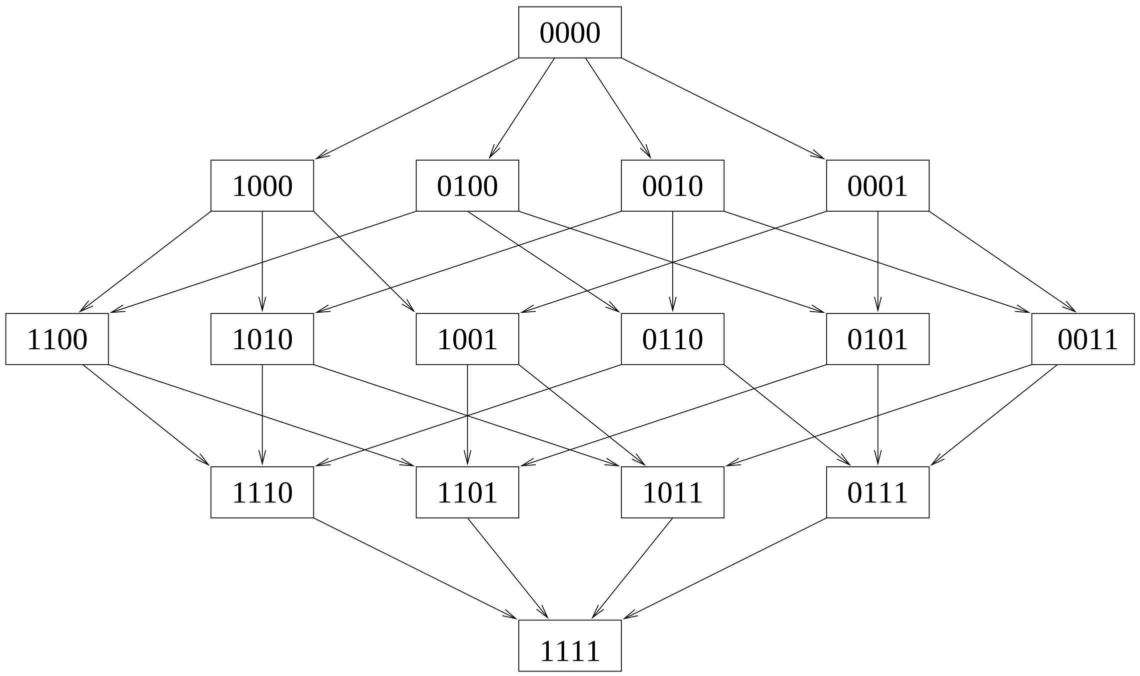
Breadth first search traversal





Peak BDD size





$$\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4$$

$$\Downarrow$$

$$\bar{x}_1\bar{x}_2(\bar{x}_3+\bar{x}_4)+(\bar{x}_1+\bar{x}_2)\bar{x}_3\bar{x}_4$$

$$\Downarrow$$

$$(\bar{x}_1+\bar{x}_2)(\bar{x}_3+\bar{x}_4)+\bar{x}_1\bar{x}_2+\bar{x}_3\bar{x}_4$$

$$\Downarrow$$

$$\bar{x}_1+\bar{x}_2+\bar{x}_3+\bar{x}_4$$

$$\Downarrow$$

$$1$$

Concurrency is structured

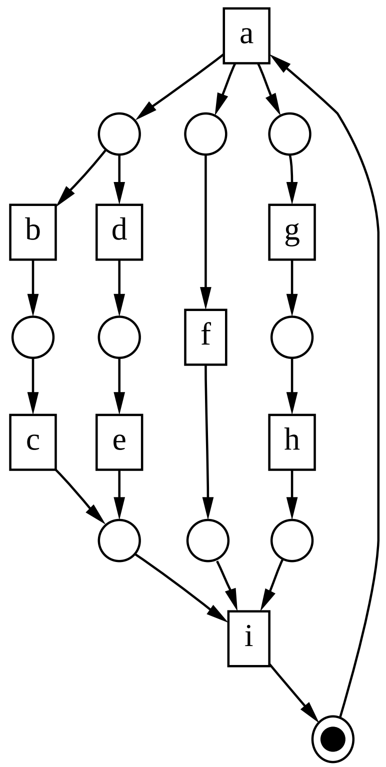
We usually have . . .

well-structured specification languages

and

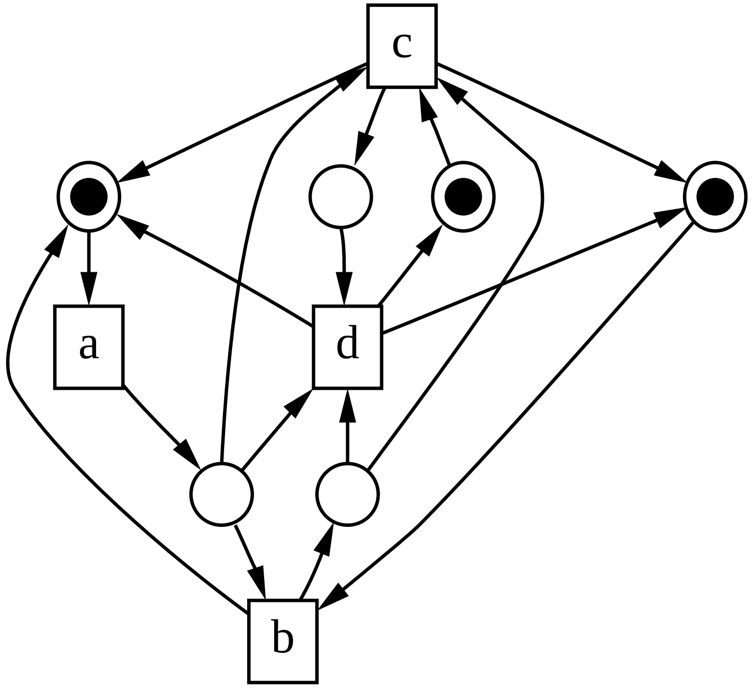
well-structured minds

begin par
P_1;
P_2;
. . .
P_n;
end par

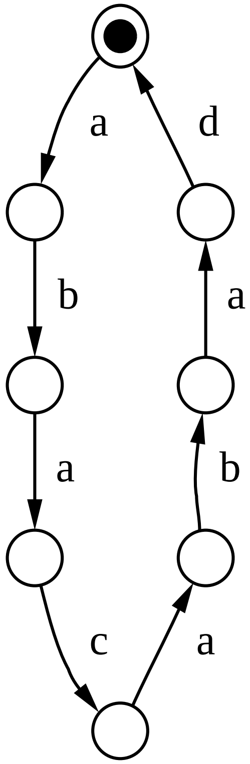


Unfoldings extract well-structured event relations

Compact representation
(6 places, 4 transitions)



Unfolded representation
(8 places, 8 transitions)



Why do we like Petri Nets ?

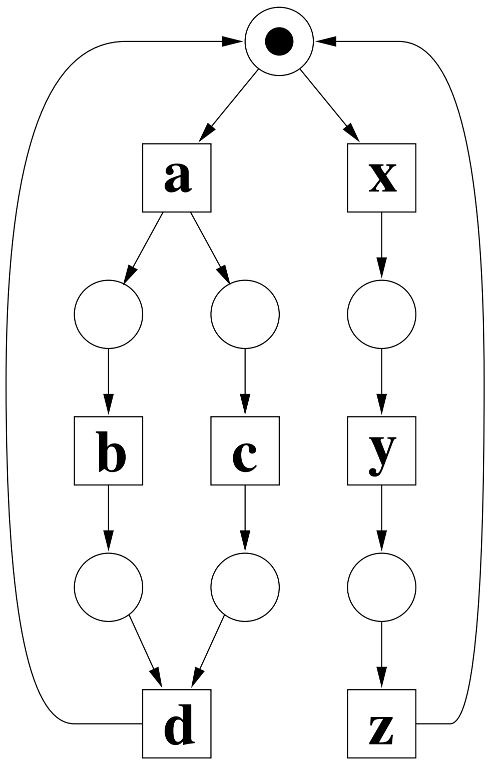
- Event-based models (CSP, CCS) can be translated into Petri nets
- There is a powerful structural theory for analysis
- Asynchronous circuits
  - Timing diagrams are isomorphic to Petri nets

Structural theory:

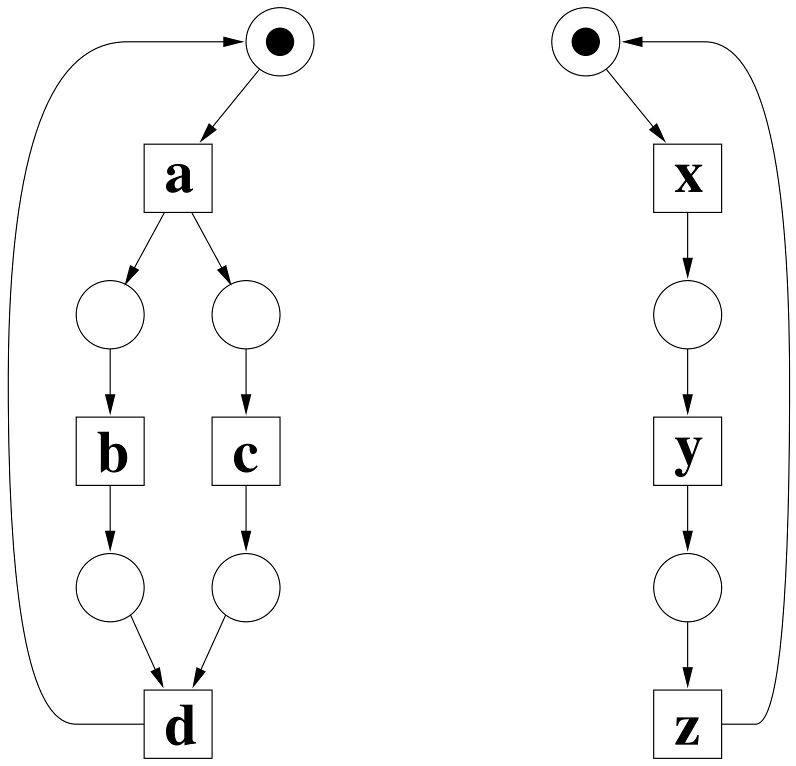
- Derives properties of the state space by analyzing the underlying graph structure of the Petri net $\Rightarrow$ polynomial algorithms
- **Drawback:** State space is approximate

What can structural theory do for us ?

$$[(a; (b \parallel c); d) \,\Box\, (x; y; z)]*$$

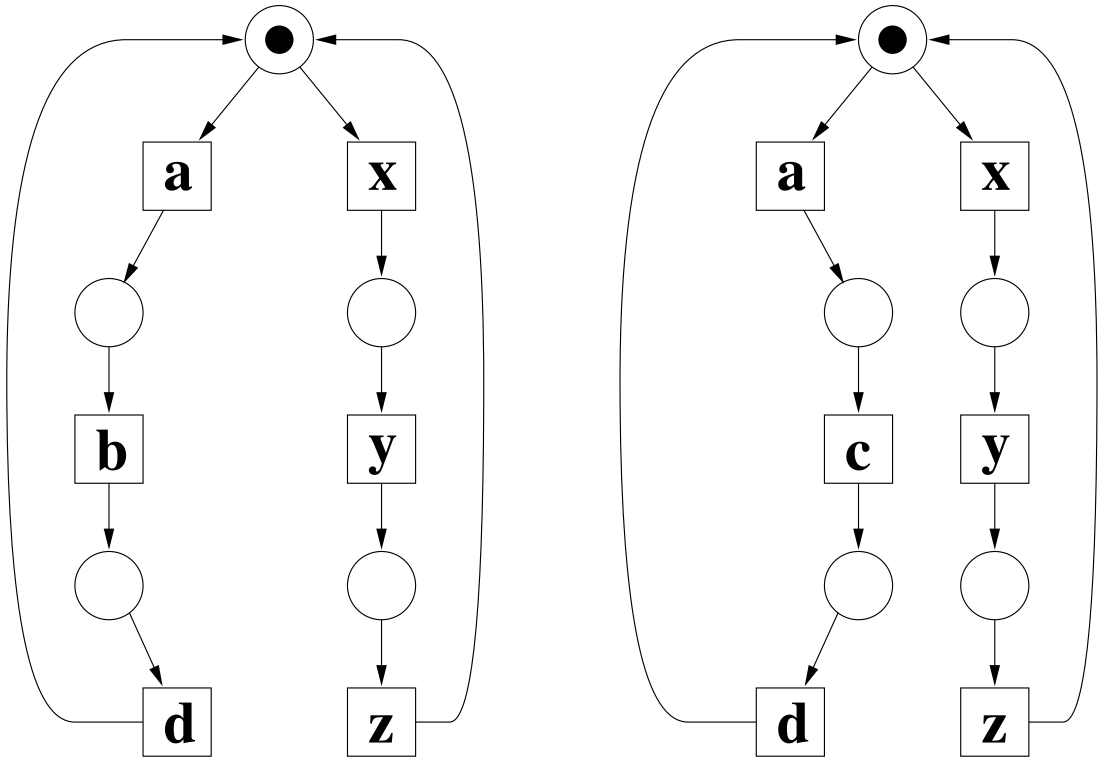


Marked-Graph decomposition



Components with concurrency (no choice) that model cyclic behavior.

State-Machine decomposition



Sequential components.

1-token SMs: 1-hot encoding $\Longrightarrow$ $\log_2$ encoding

Structural theory

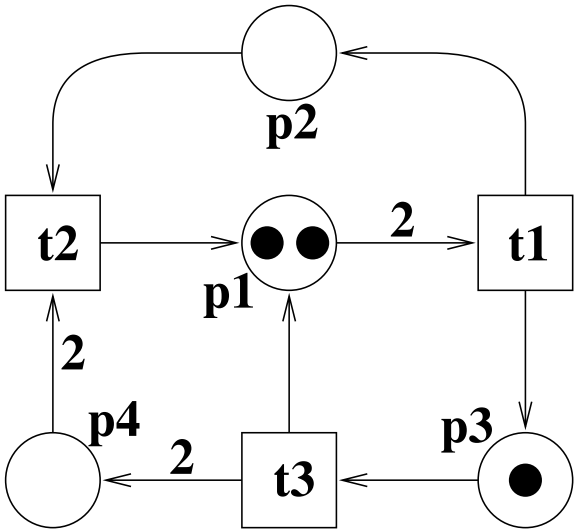
State Machines and Marked Graphs:

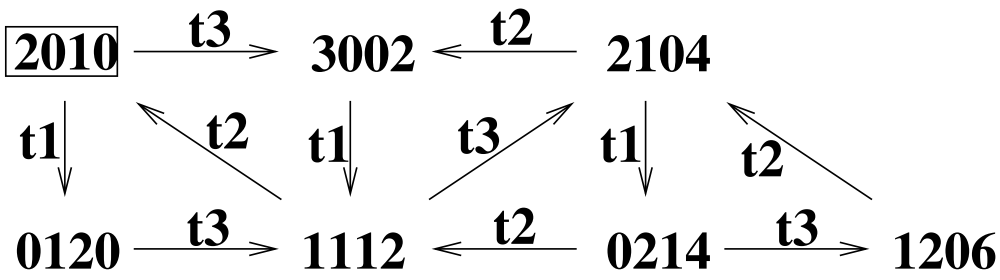
- well-structured objects
- can be composed to derive more complex structures with concurrency and choice

Particular case of more general concepts:

- place-invariants (conservative behavior)
- transition-invariants (repetitive behavior)

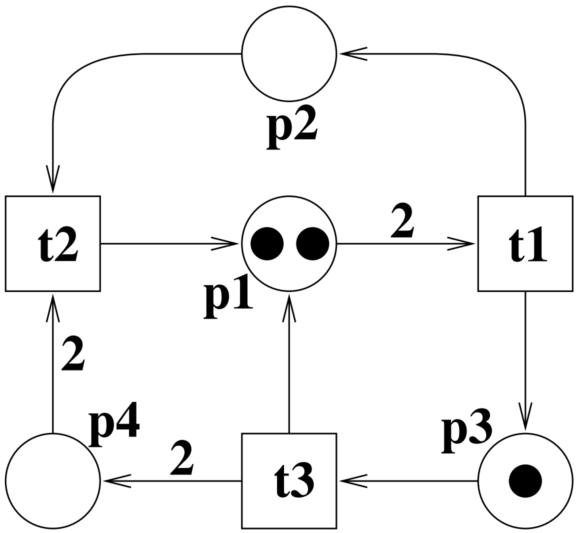
Reachability space





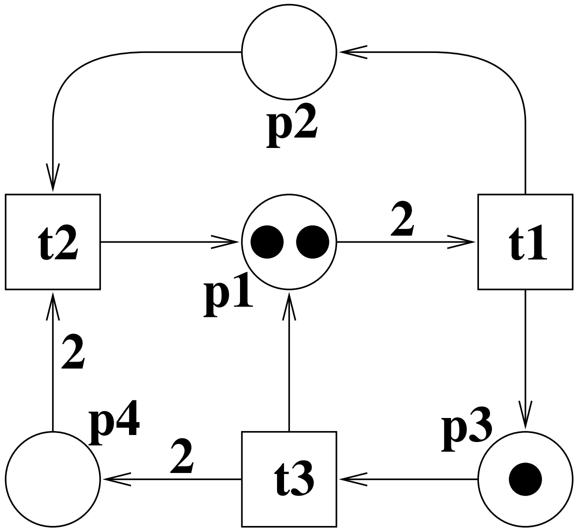
from [Murata, IEEE Proc., 1989]

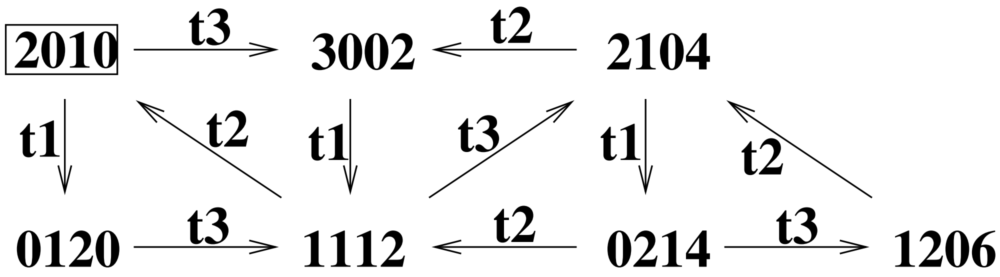
Incidence matrix of a Petri net



$$
\begin{array}{c|c}
 & \begin{array}{ccc} t_1 & t_2 & t_3 \end{array} \\
\begin{array}{c} p_1 \\ p_2 \\ p_3 \\ p_4 \end{array} & \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix}
\end{array}
$$

Marking equation

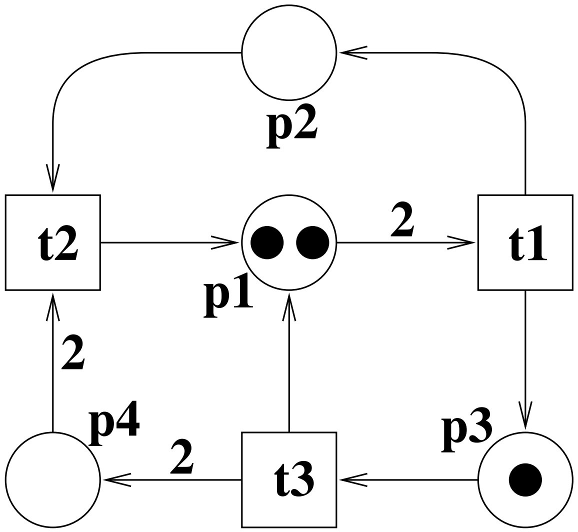


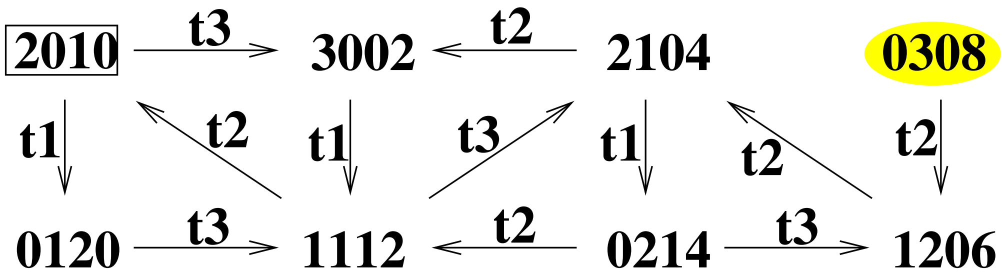


$$\begin{bmatrix} 2 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \\ 4 \end{bmatrix}$$

$$M_i + A^T u = M_j$$

Approximating the state space



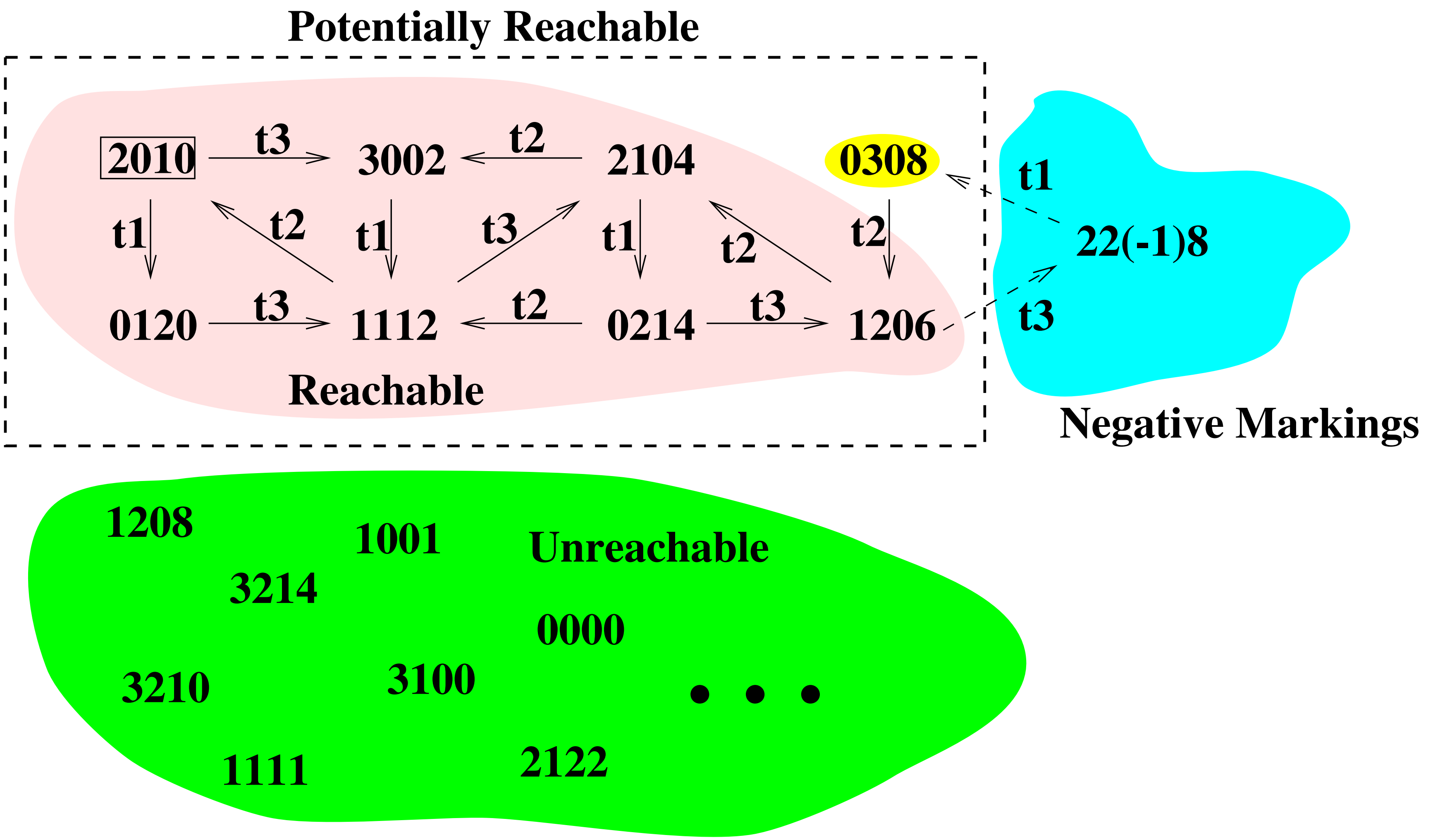


$$M_i \text{ reachable from } M_0 \Longrightarrow \exists u \geq 0 : M_i = M_0 + A^T u$$

but . . .

$$\begin{bmatrix} 1 \\ 2 \\ 0 \\ 6 \end{bmatrix} + \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 0 \\ 8 \end{bmatrix}$$

Taxonomy of the state space



Marking equation and invariants

Place invariant: $Ay = 0$ y is a place-vector

Basis of all P- and T-invariants:

- Smith Normal Form of the Incidence Matrix (Desel et al., TCS 1996)

$$\begin{aligned} 2p_1 + 4p_2 - p_4 &= 4 \\ p_1 + p_2 + p_3 &= 3 \end{aligned}$$

Structural bounds by LP:

$\max(p_1) = \max(p_2) = 3$ $\max(p_3) = 2$ $\max(p_4) = 8$

Same power as marking equation.

and now the BRUTE FORCE !!!

but the force is not so brute ...

- each invariant characterizes a set of states
- the *conjunction* of invariants characterizes the *potentially reachable* state space
- **ONLY** potentially reachable states must be encoded
- (Algebraic) decision diagrams can easily manipulate linear expressions

Encoding invariants (I)

$$\boxed{2p_1 + 4p_2 - p_4 = 4}$$

$$\text{max}(p_1) = \text{max}(p_2) = 3 \qquad \text{max}(p_4) = 8$$

We can use complementary places: $p_4 + \overline{p_4} = 8$

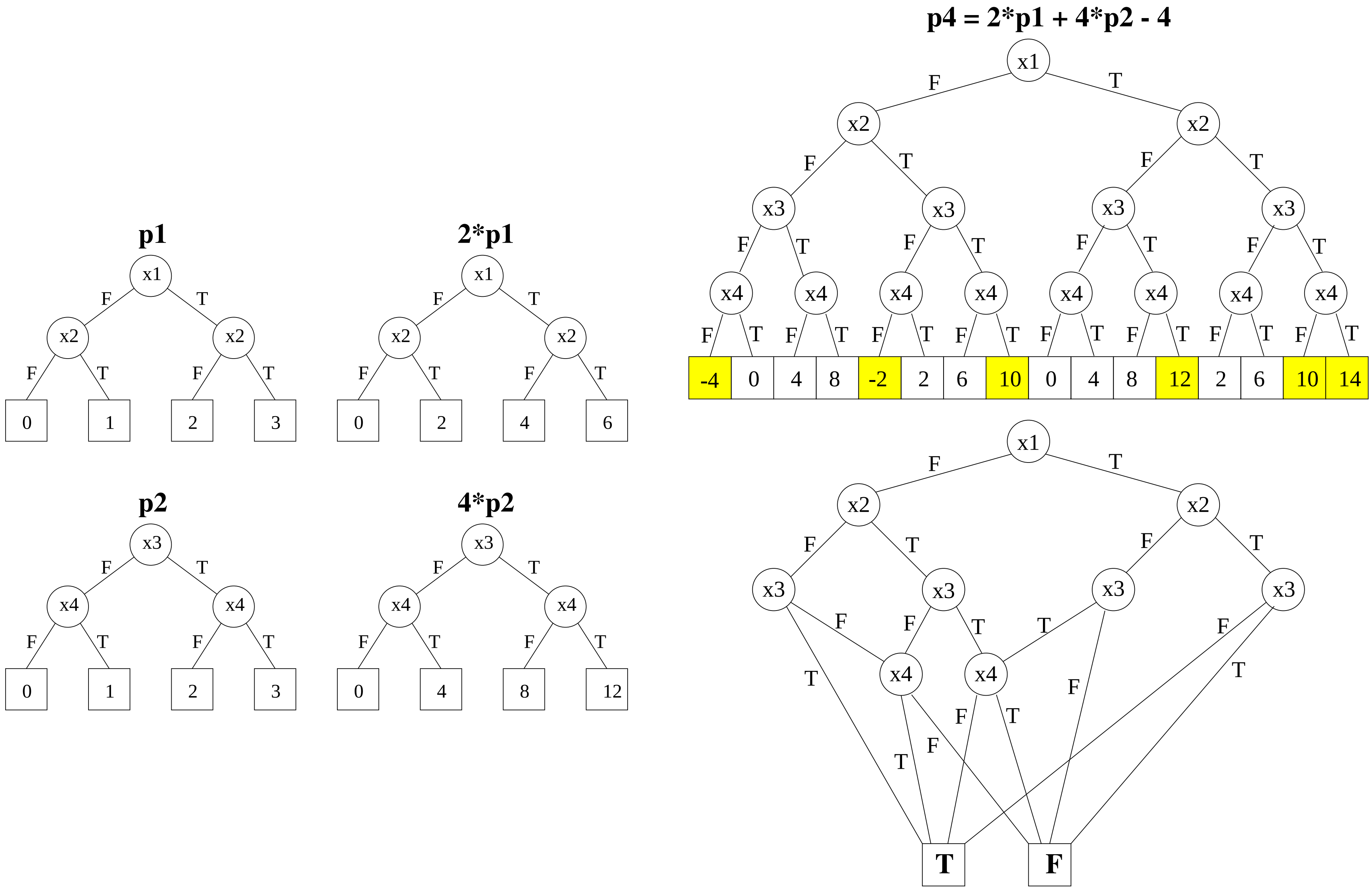
$$\boxed{2p_1 + 4p_2 + \overline{p_4} = 12}$$

The number of solutions of

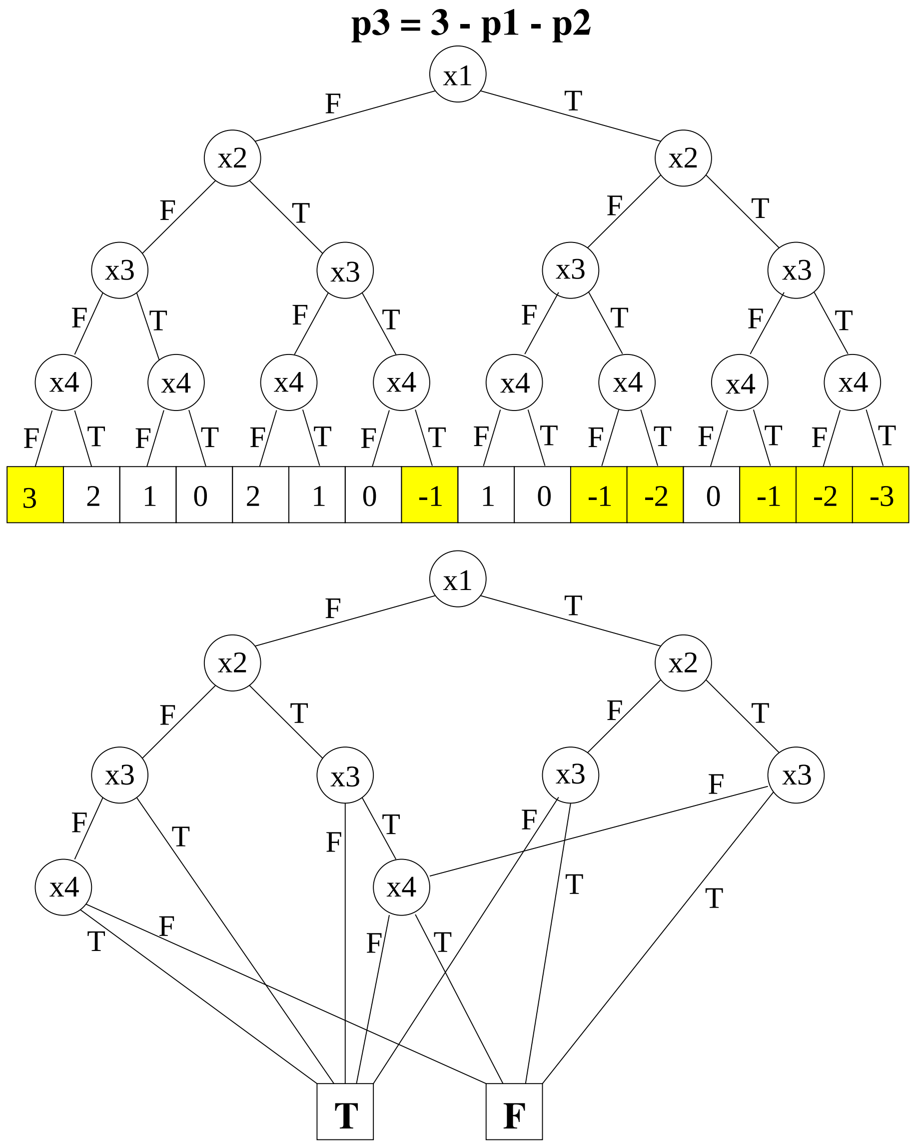
$$a_1 x_1 + a_2 x_2 + \ldots + a_k x_k = n$$

is approx. cn^{t-1} where c is the coefficient of $(1 - x)^{-n}$

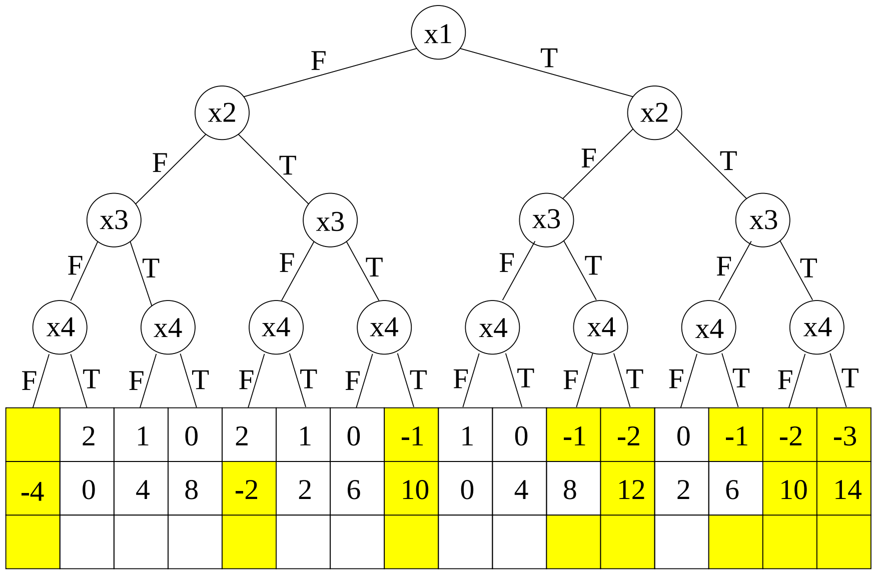
Encoding invariants (II)

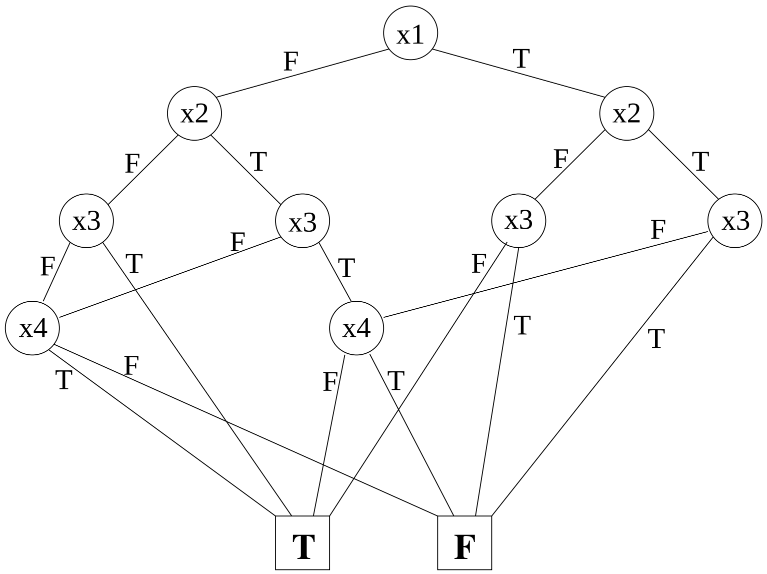


Encoding invariants (III)

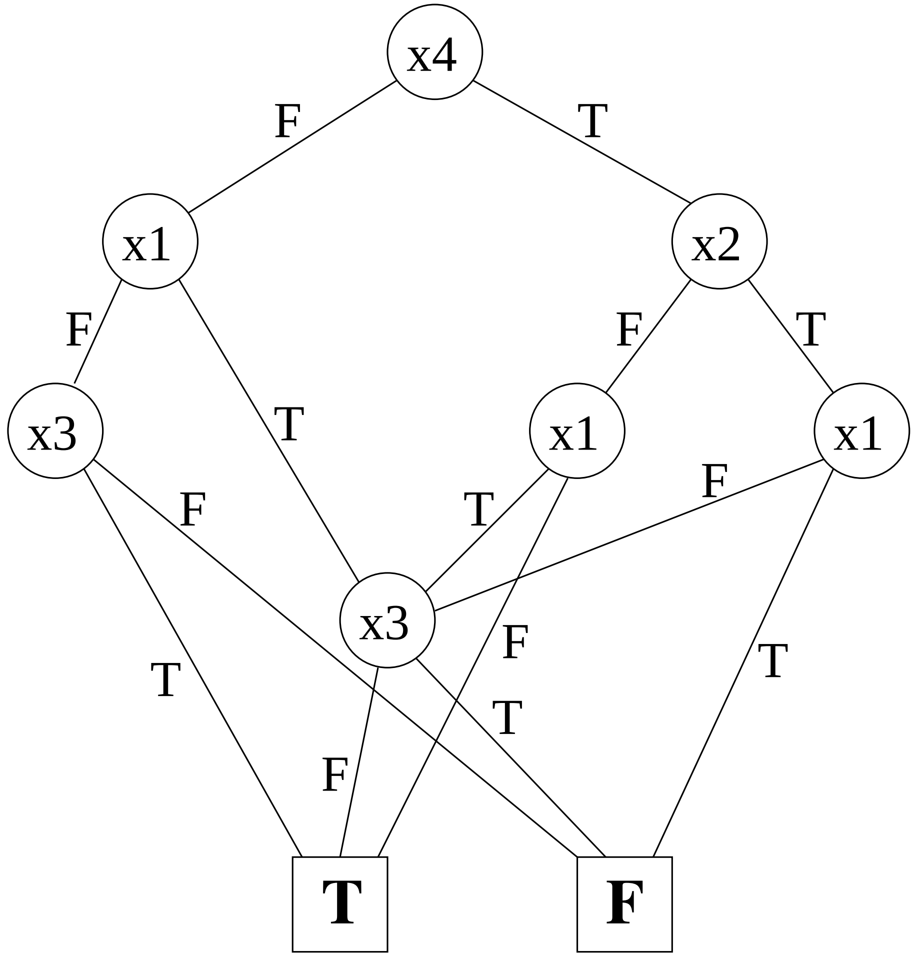


Encoding invariants (IV)

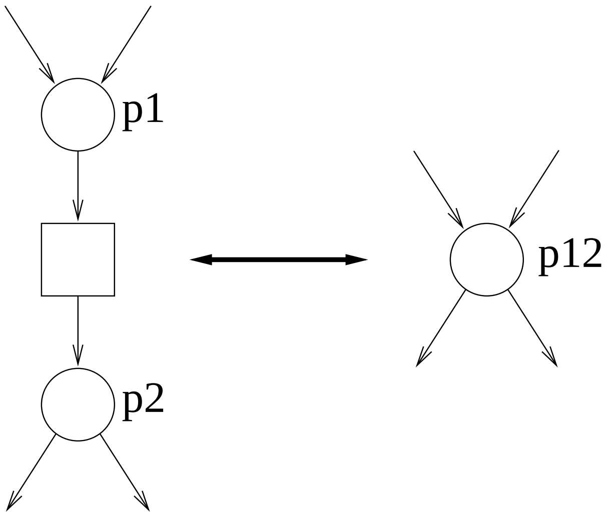




Encoding invariants (V)



Reduction of series places



$$R = R^{\bullet} p_{12} + R^{\circ} \overline{p_{12}}$$

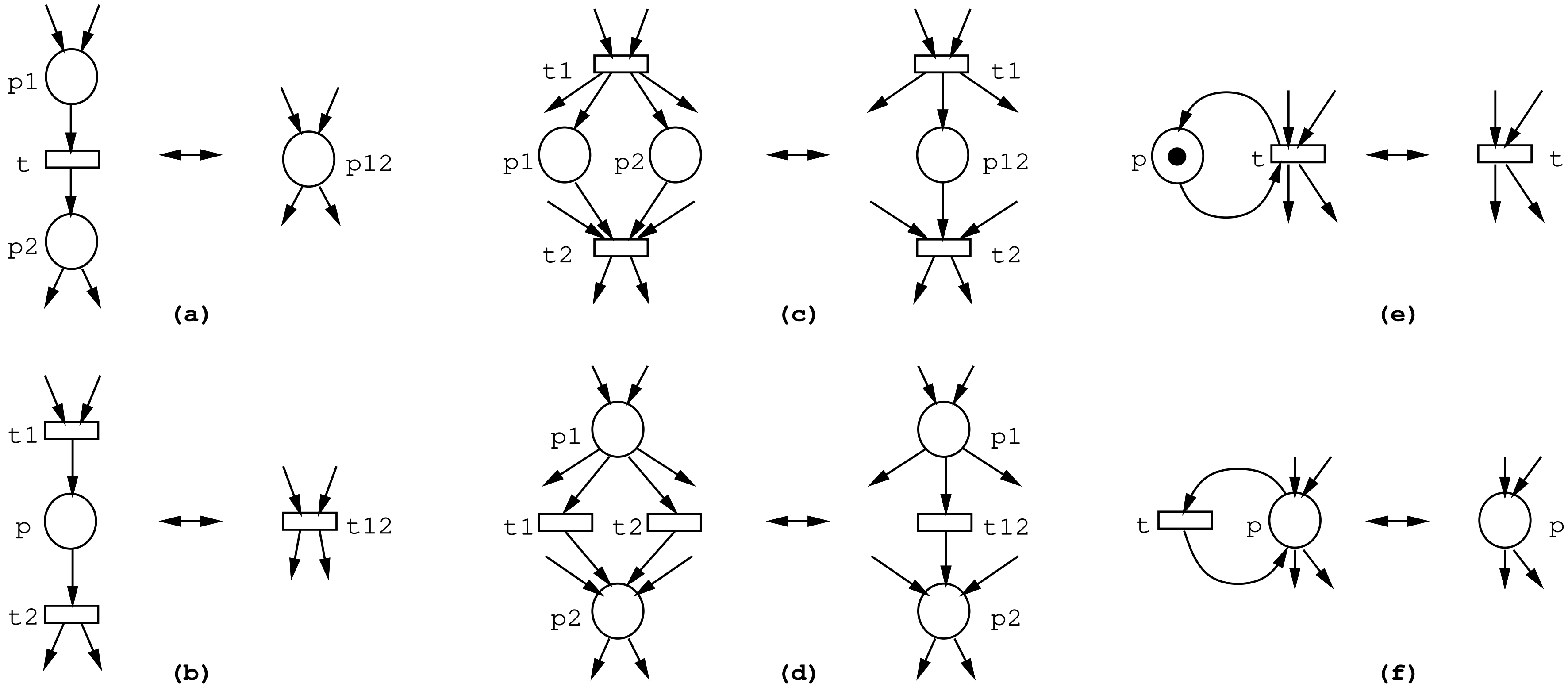
$$\begin{array}{lcl} p_{12} & \Rightarrow & p_1 \text{ or } p_2 \text{ marked} \\ \overline{p_{12}} & \Rightarrow & p_1 \text{ and } p_2 \text{ not marked} \end{array}$$

Substitute p_{12} by two variables: x and y

$$\begin{array}{rcl} x & \Rightarrow & p_1\overline{p_2} + \overline{p_1}p_2 \\ x & \Rightarrow & \overline{p_1}\ \overline{p_2} \\ xy & \Rightarrow & p_1 \\ x\overline{y} & \Rightarrow & p_2 \end{array}$$

$$R_{new} = R^{\bullet}(xy + x\overline{y}) + R^{\circ}\overline{x} = R[x \leftarrow p_{12}]$$

Local reductions



Computing the state space by reductions

Reduce Petri net (series/parallel/self-loop reductions)

$$\Downarrow$$

Encode and calculate the state space (BFS traversal)

$$\Downarrow$$

Re-encode back to the originial Petri net (same BDD size)

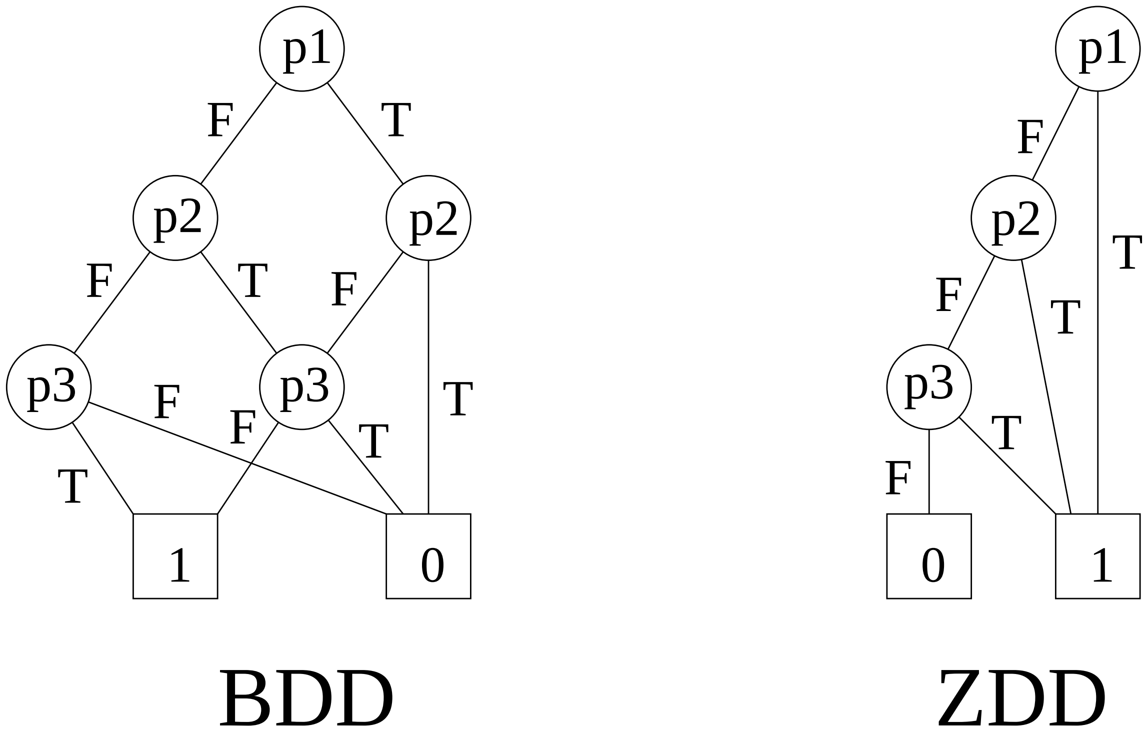
- **For safe series/parallel Petri nets:** reduction to only one marked place

$$R = 1$$

Sparse vs. Dense encoding

		Sparse			Dense		
example	markings	vars	BDD	CPU	vars	BDD	CPU
muller-30	6.0×10^7	120	4475	585	60	1315	32
muller-40	4.6×10^{10}	150	4897	7046	80	2339	131
muller-50	3.6×10^{13}	200	-	-	100	3651	449
phil-5	8.5×10^4	65	640	2	35	155	3
phil-8	7.8×10^7	104	2933	12	56	373	19
phil-10	7.4×10^9	130	1689	90	70	425	285
slot-5	1.7×10^6	50	492	14	25	131	5
slot-7	7.9×10^8	70	807	109	35	239	9
slot-9	3.8×10^{11}	90	-	-	45	400	110

Zero-Supressed BDDs [Minato 93]

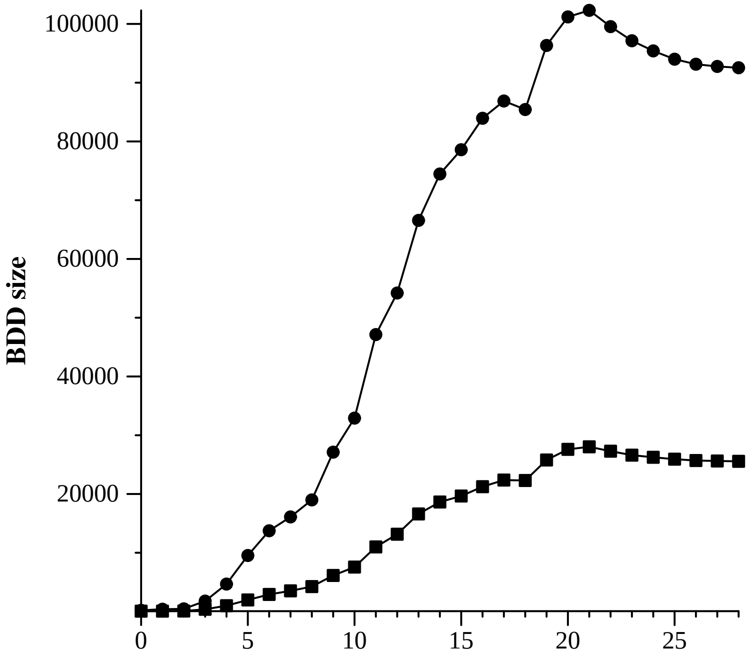


$$p1 + p2 + p3 = 1$$

ZDDs [Yoneda et al. 1996] vs. BDDs

example	markings	vars	ZDD	vars	BDD
DMEspec8	7.8×10^5	137	32178	85	1748
DMEspec9	3.5×10^6	154	71602	94	2544
DMEcir5	8.5×10^5	491	92214	249	47952
DMEcir7	9.0×10^7	687	504324	347	394334
JJreg-a	1.8×10^6	251	952246	122	17874
JJreg-b	1.1×10^5	248	181701	120	24355

Sparse/Dense encoding: example



220 places, 1570 transitions

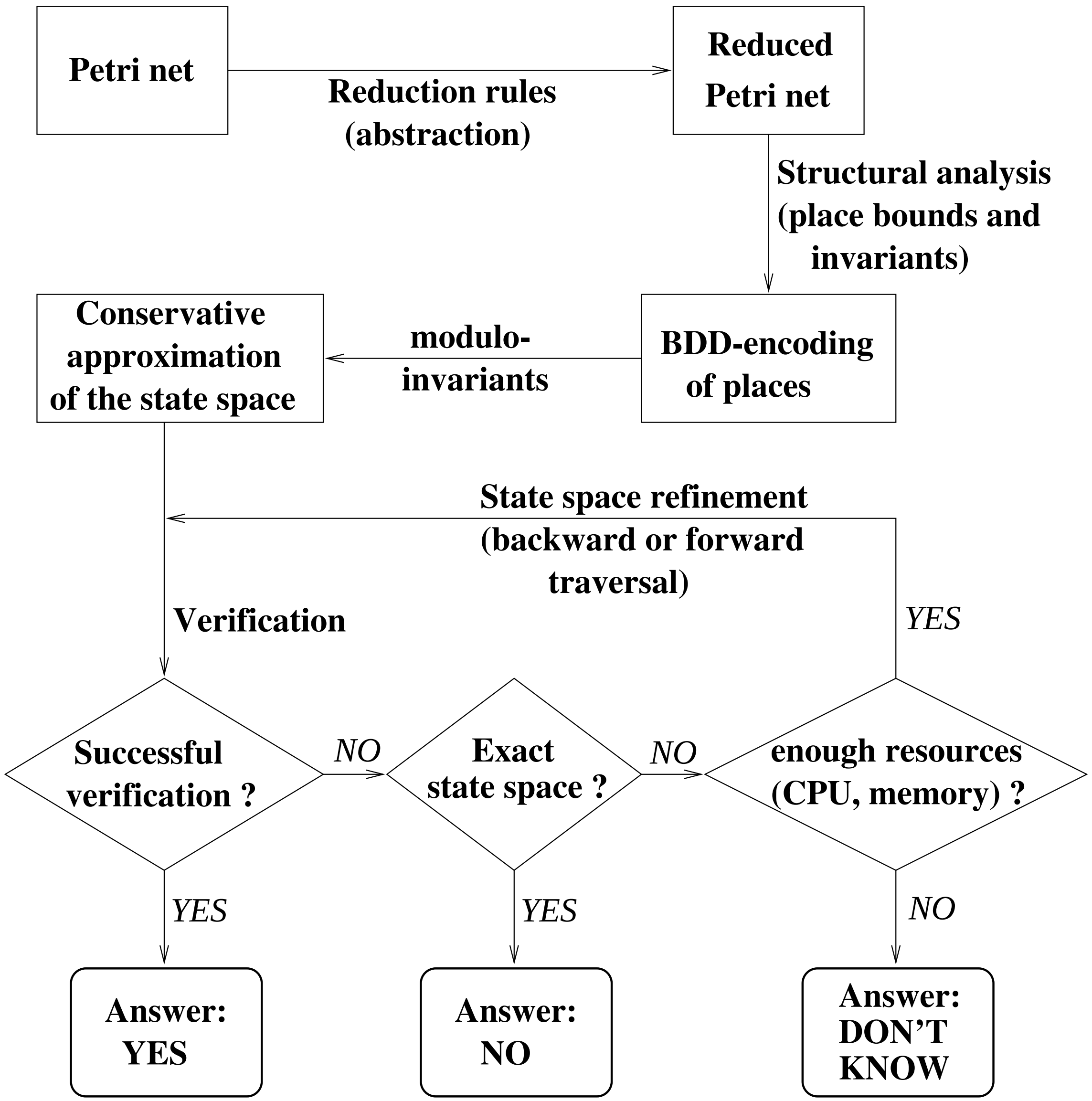
Potentially RMs (PRM): 4.5×10^{14}
RMs: 2×10^{7}

$$2^{204} = 2.6 \times 10^{61}$$
$$2^{58} = 2.9 \times 10^{17}$$

Min number of vars for PRM: 49

	Sparse	Dense	PRM
Variables	204	58	58
BDD size	92543	25562	59
CPU (secs)	1003	171	10

Putting everything together



Conclusions

- The structure of a concurrent provides a significant amount of information on its state space
- Structural information is not enough to derive the exact state space
- BDDs are not the panacea (only the very last resort)
- There is still a lot to be done
  - Other structural objects: siphons, traps
  - Structural concurrency relation
  - Partial-order reductions
  - Constrained Logic Programming (CLP)
  - Variable ordering