Saturation Enhanced with Conditional Locality: Application to Petri Nets

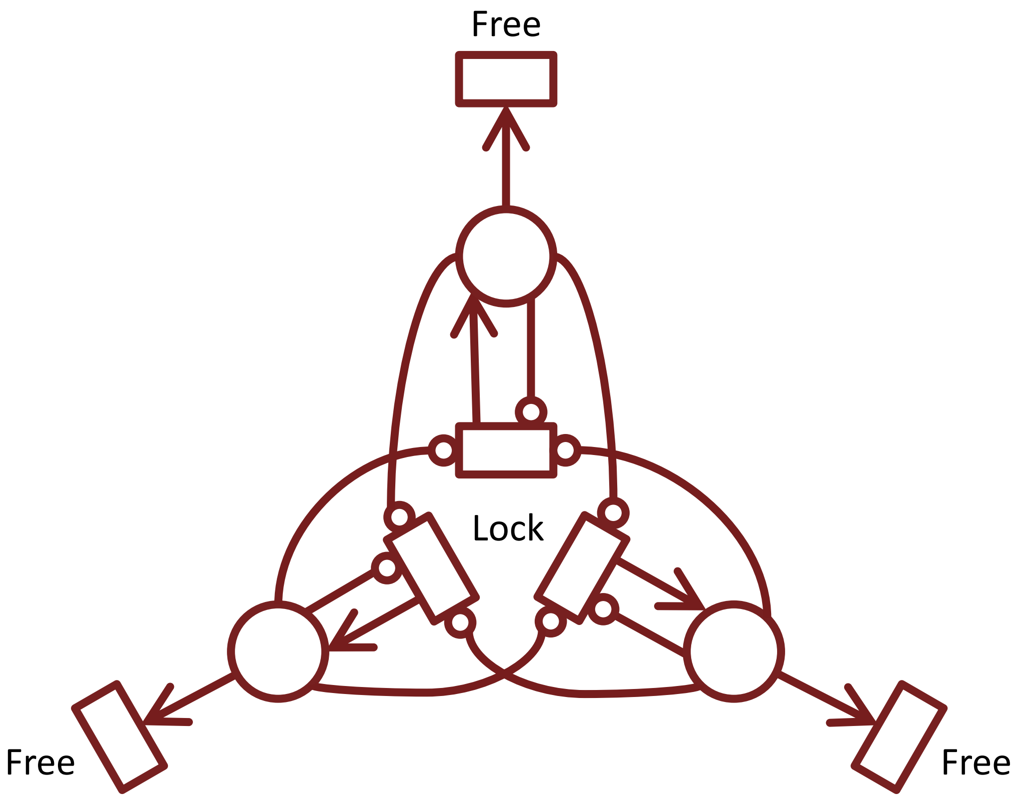
Vince Molnár[1,2], István Majzik[1]

{**molnarv**,majzik}@mit.bme.hu

[1] Budapest University of Technology and Economics | Department of Measurement and Information Systems | Fault Tolerant Systems Research Group
[2] Hungarian Academy of Sciences | MTA-BME Lendület Research Group on Cyber-Physical Systems

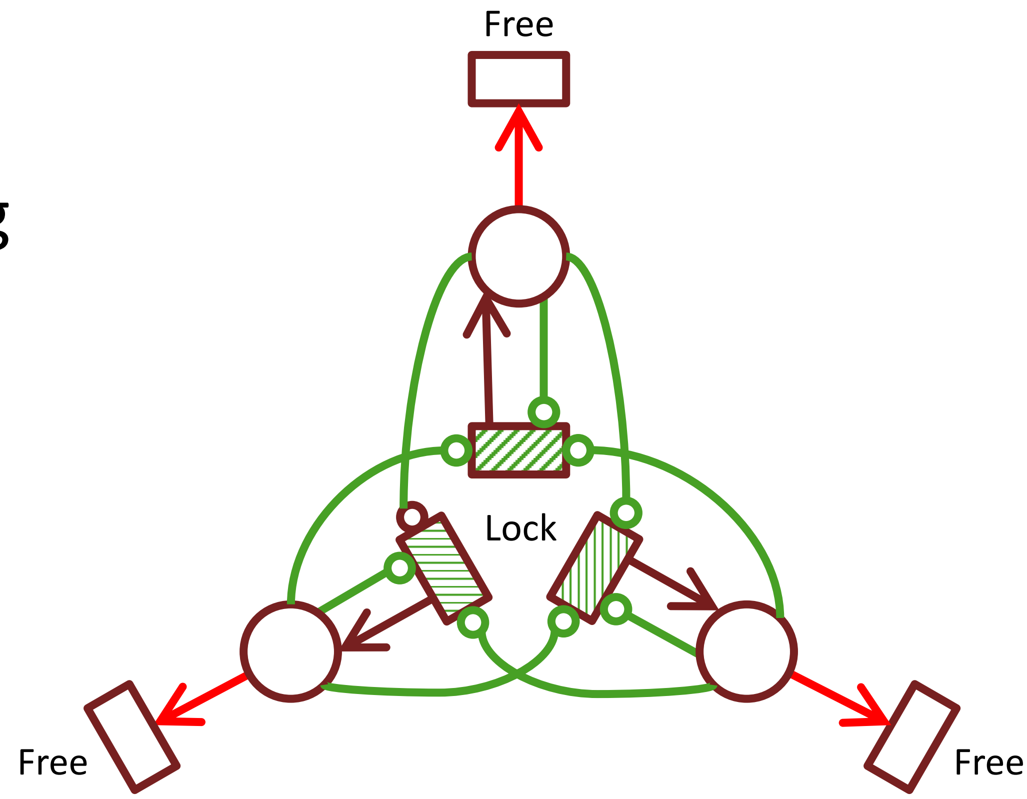
Introduction

- Model checking
  - State space exploration
  - Property analysis
- Symbolic model checking
  - Characteristic function
  - Decision diagrams
  - Saturation algorithm
- In this paper…
  - Conditional locality
  - General representations
  - Enhanced saturation effect



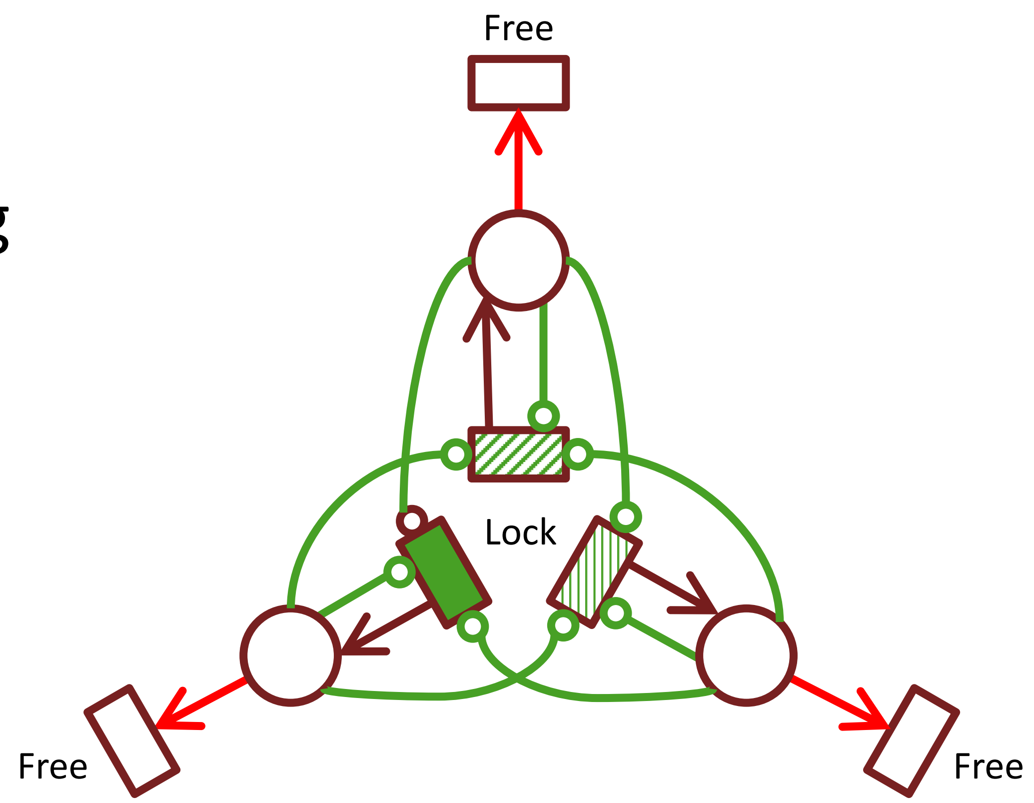
Introduction

- Model checking
  - State space exploration
  - Property analysis
- Symbolic model checking
  - Characteristic function
  - Decision diagrams
  - Saturation algorithm
- In this paper…
  - Conditional locality
  - General representations
  - Enhanced saturation effect



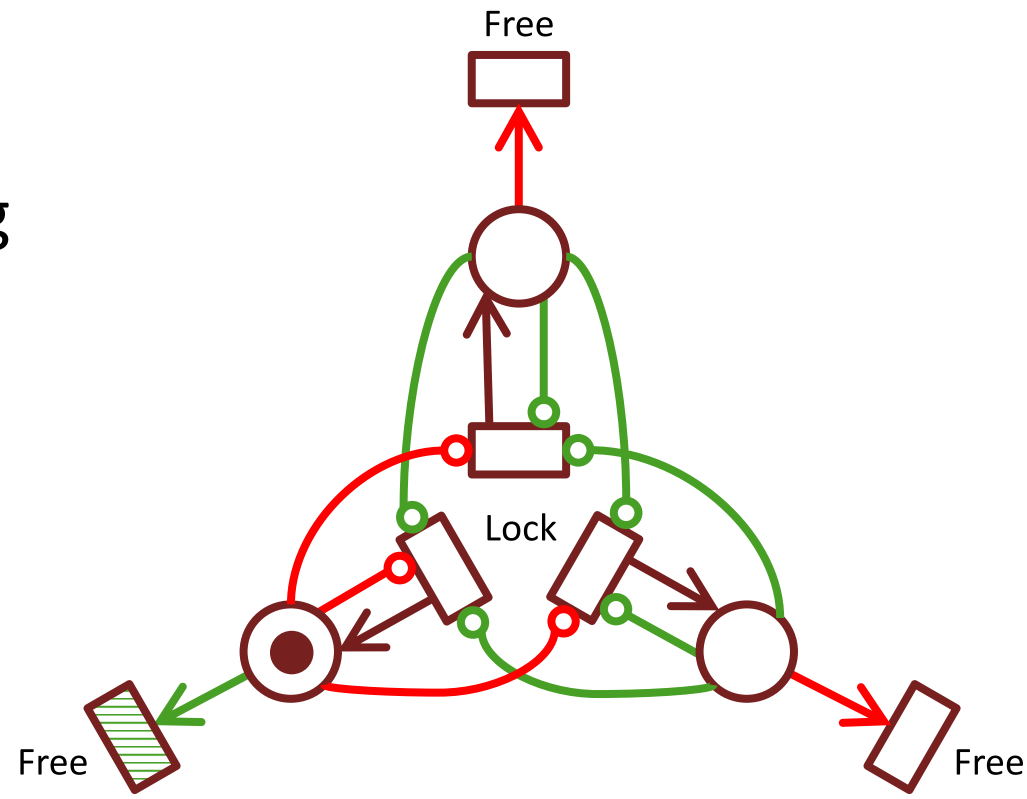
Introduction

- Model checking
  - State space exploration
  - Property analysis
- Symbolic model checking
  - Characteristic function
  - Decision diagrams
  - Saturation algorithm
- In this paper…
  - Conditional locality
  - General representations
  - Enhanced saturation effect



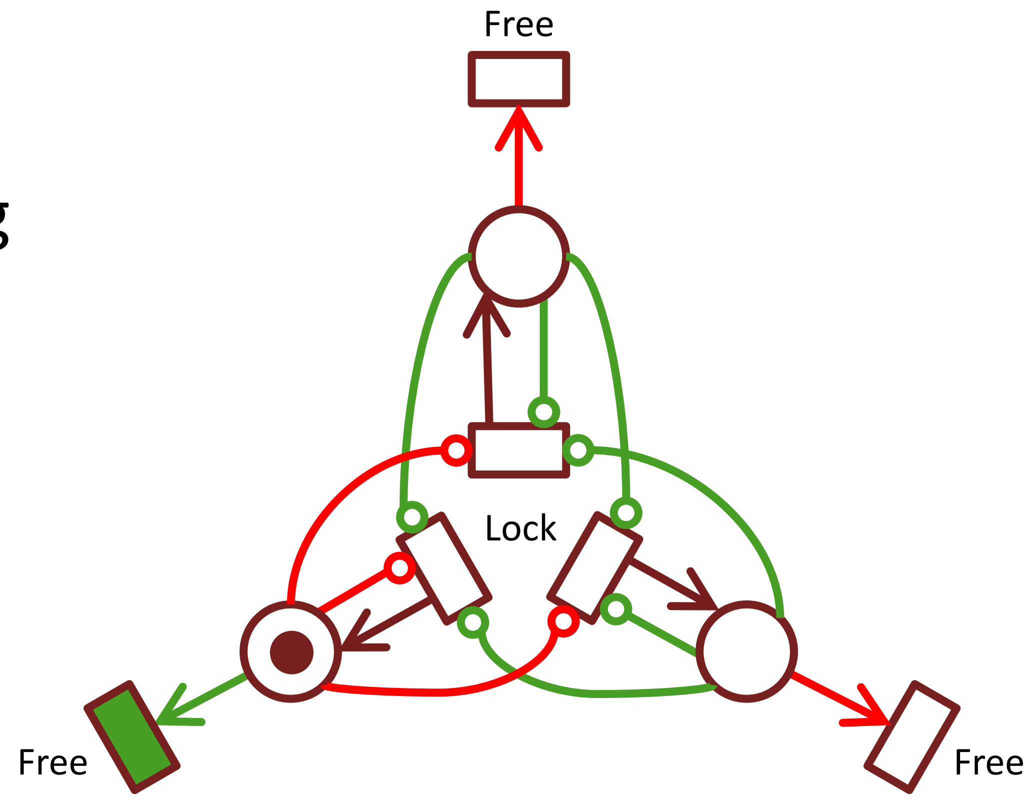
Introduction

- Model checking
  - State space exploration
  - Property analysis
- Symbolic model checking
  - Characteristic function
  - Decision diagrams
  - Saturation algorithm
- In this paper…
  - Conditional locality
  - General representations
  - Enhanced saturation effect



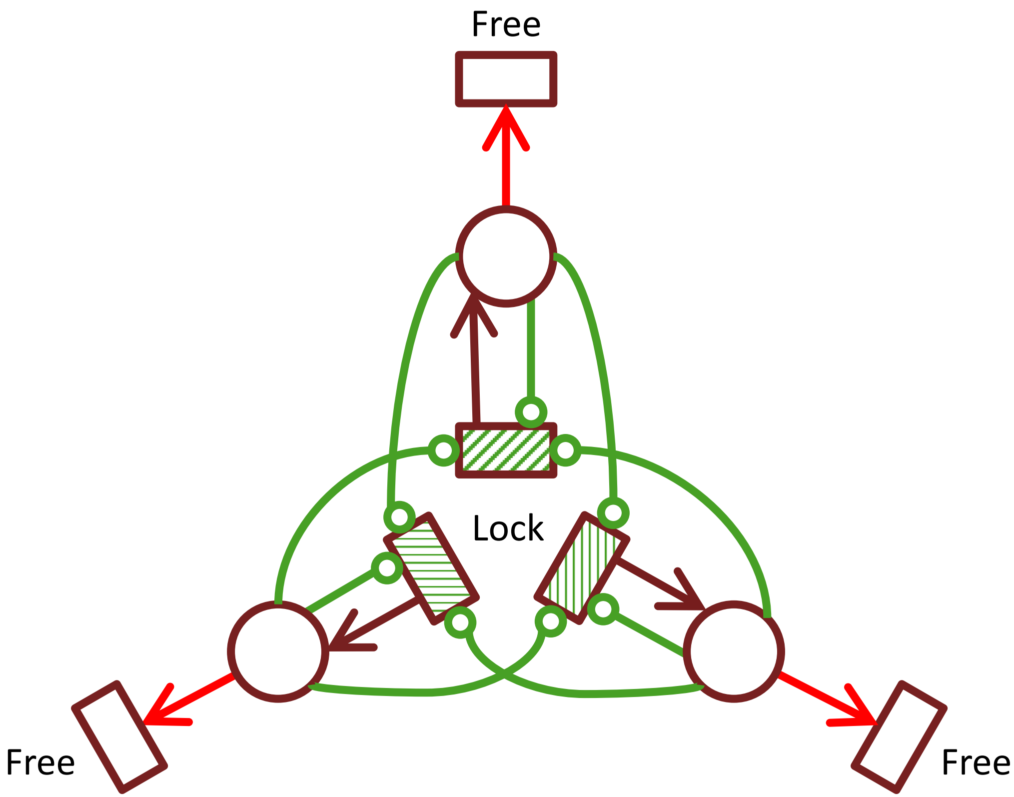
Introduction

- Model checking
  - State space exploration
  - Property analysis
- Symbolic model checking
  - Characteristic function
  - Decision diagrams
  - Saturation algorithm
- In this paper…
  - Conditional locality
  - General representations
  - Enhanced saturation effect



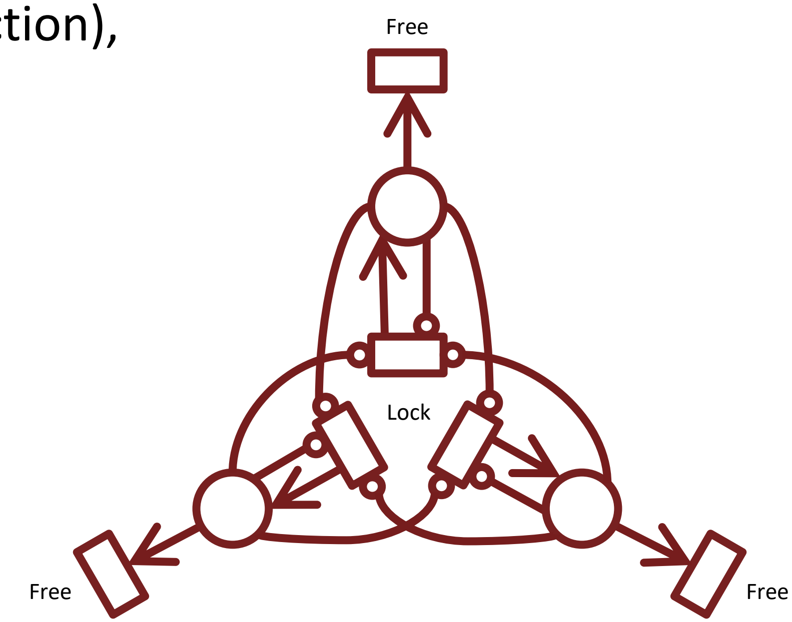
Introduction

- Model checking
  - State space exploration
  - Property analysis
- Symbolic model checking
  - Characteristic function
  - Decision diagrams
  - Saturation algorithm
- In this paper…
  - Conditional locality
  - General representations
  - Enhanced saturation effect



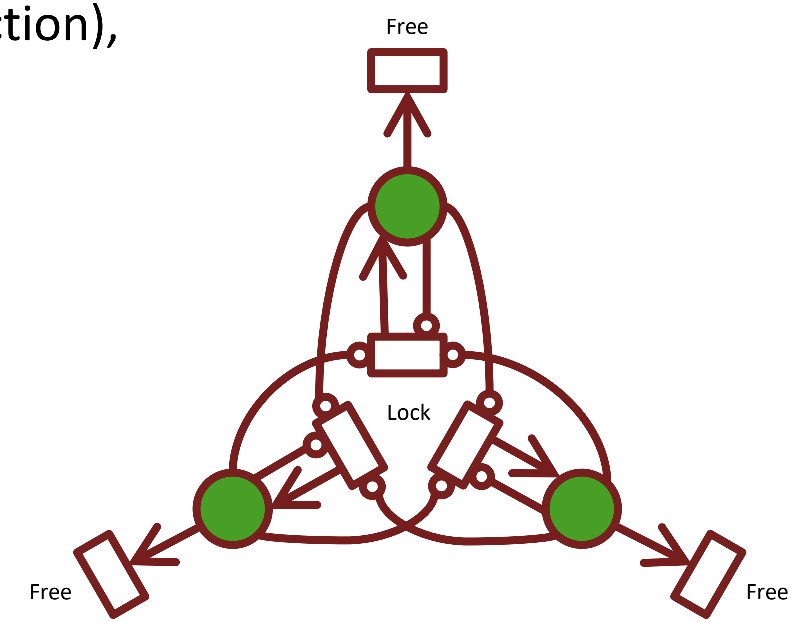
Partitioned Transition System

- Low-level formalism that preserves structure of high-level model
- A **partitioned transition system (PTS)** is a tuple $(V, D, S^0, \mathcal{E}, \mathcal{N})$ s.t.:
  - V is the set of **variables**
  - D is the **domain** function ($D(x_k) \subseteq \mathbb{N}$ for all $x_k \in V$)
  - $S^0 \subseteq \hat{S}$ is the set of **initial states** ($\hat{S}$ is the potential state space)
  - $\mathcal{E}$ is the set of high-level **events**
  - $\mathcal{N} \subseteq \hat{S} \times \hat{S}$ is the **next-state relation** (function), partitioned by $\mathcal{E}$ such that $\mathcal{N} = \bigcup_{\alpha\in\mathcal{E}} \mathcal{N}_\alpha$
- In Petri nets:



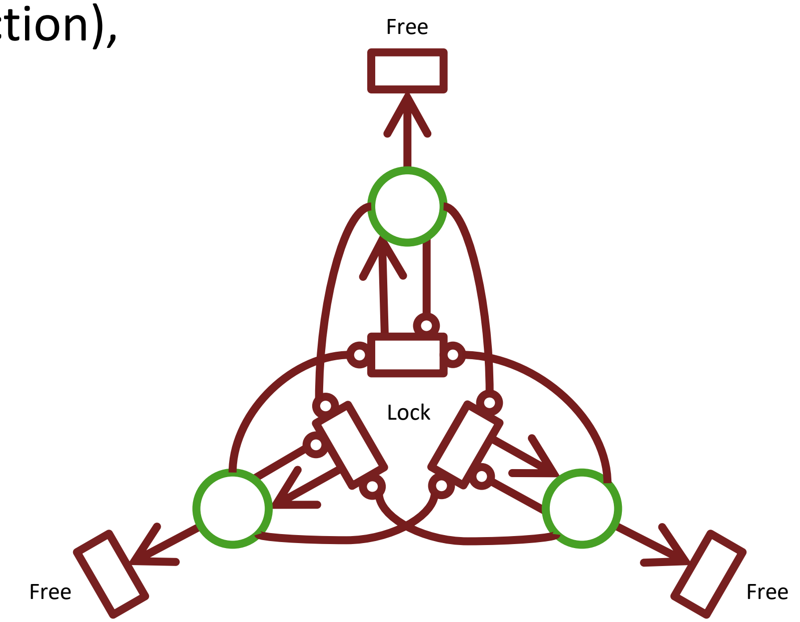
Partitioned Transition System

- Low-level formalism that preserves structure of high-level model
- A **partitioned transition system (PTS)** is a tuple $(V, D, S^0, \mathcal{E}, \mathcal{N})$ s.t.:
  - V is the set of **variables**
  - D is the **domain** function ($D(x_k) \subseteq \mathbb{N}$ for all $x_k \in V$)
  - $S^0 \subseteq \hat{S}$ is the set of **initial states** ($\hat{S}$ is the potential state space)
  - $\mathcal{E}$ is the set of high-level **events**
  - $\mathcal{N} \subseteq \hat{S} \times \hat{S}$ is the **next-state relation** (function), partitioned by $\mathcal{E}$ such that $\mathcal{N} = \bigcup_{\alpha \in \mathcal{E}} \mathcal{N}_\alpha$
- In Petri nets:
  - Variables are **places** (domain is $\mathbb{N}$)



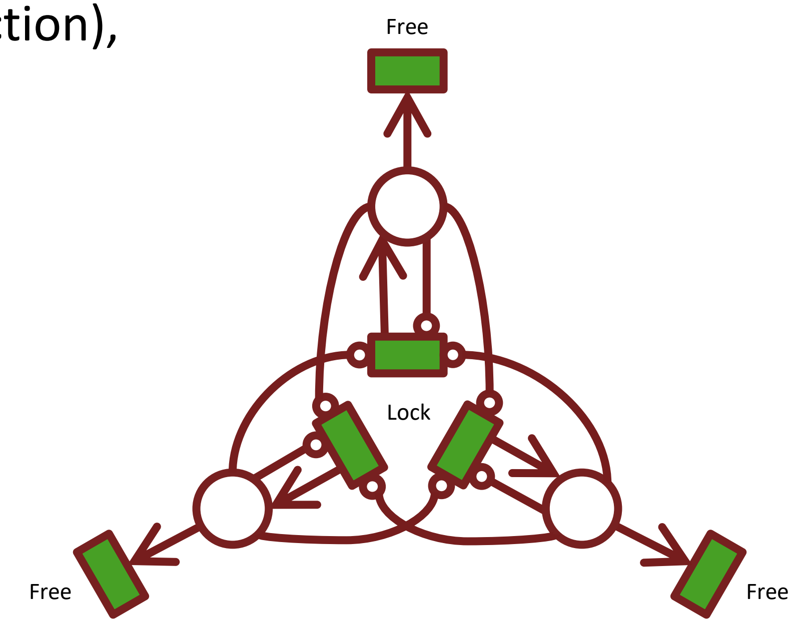
Partitioned Transition System

- Low-level formalism that preserves structure of high-level model
- A **partitioned transition system (PTS)** is a tuple $(V, D, S^0, \mathcal{E}, \mathcal{N})$ s.t.:
  - V is the set of **variables**
  - D is the **domain** function ($D(x_k) \subseteq \mathbb{N}$ for all $x_k \in V$)
  - $S^0 \subseteq \hat{S}$ is the set of **initial states** ($\hat{S}$ is the potential state space)
  - $\mathcal{E}$ is the set of high-level **events**
  - $\mathcal{N} \subseteq \hat{S} \times \hat{S}$ is the **next-state relation** (function), partitioned by $\mathcal{E}$ such that $\mathcal{N} = \bigcup_{\alpha \in \mathcal{E}} \mathcal{N}_\alpha$
- In Petri nets:
  - Variables are **places** (domain is $\mathbb{N}$)
  - Initial state is **initial marking**



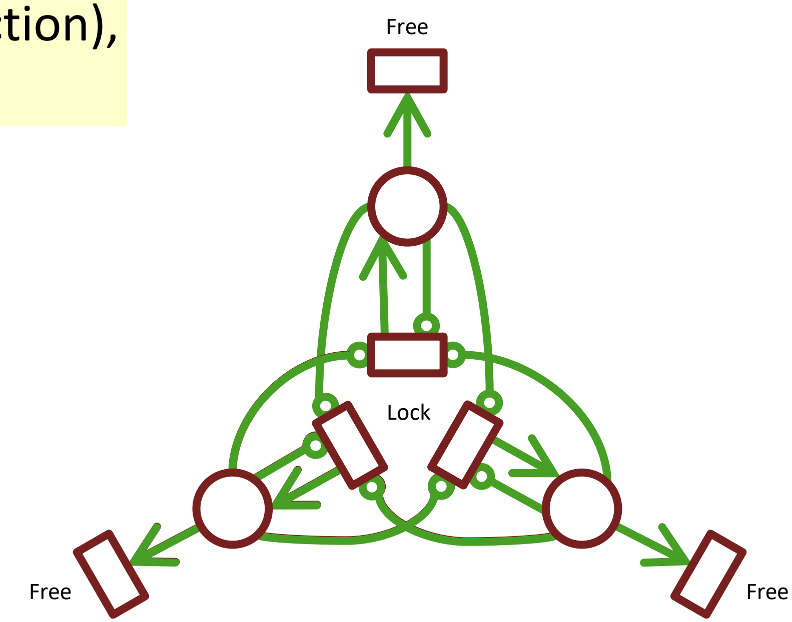
Partitioned Transition System

- Low-level formalism that preserves structure of high-level model
- A **partitioned transition system (PTS)** is a tuple $(V, D, S^0, \mathcal{E}, \mathcal{N})$ s.t.:
  - V is the set of **variables**
  - D is the **domain** function ($D(x_k) \subseteq \mathbb{N}$ for all $x_k \in V$)
  - $S^0 \subseteq \hat{S}$ is the set of **initial states** ($\hat{S}$ is the potential state space)
  - $\mathcal{E}$ is the set of high-level **events**
  - $\mathcal{N} \subseteq \hat{S} \times \hat{S}$ is the **next-state relation** (function), partitioned by $\mathcal{E}$ such that $\mathcal{N} = \bigcup_{\alpha \in \mathcal{E}} \mathcal{N}_\alpha$
- In Petri nets:
  - Variables are **places** (domain is $\mathbb{N}$)
  - Initial state is **initial marking**
  - Events are **transitions**



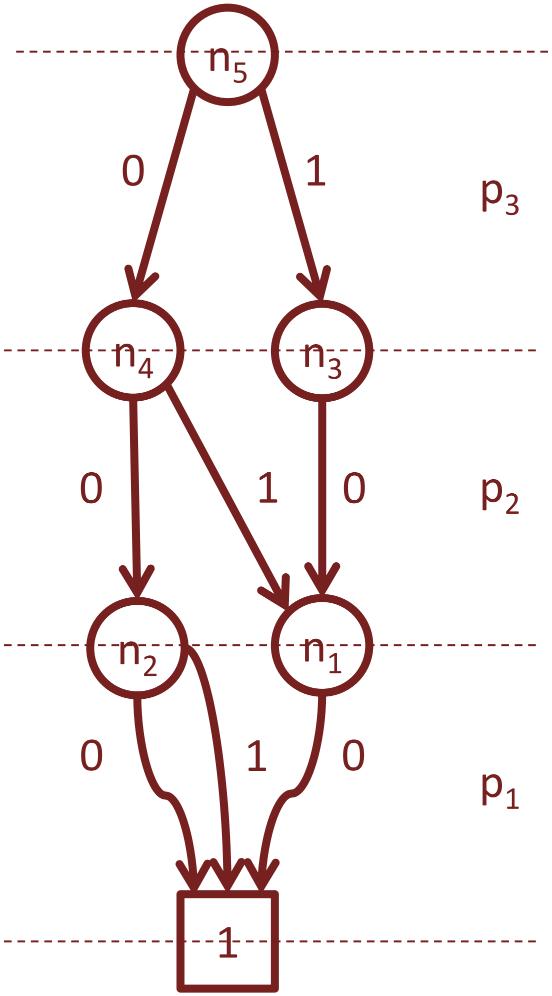
Partitioned Transition System

- Low-level formalism that preserves structure of high-level model
- A **partitioned transition system (PTS)** is a tuple $(V, D, S^0, \mathcal{E}, \mathcal{N})$ s.t.:
  - V is the set of **variables**
  - D is the **domain** function ($D(x_k) \subseteq \mathbb{N}$ for all $x_k \in V$)
  - $S^0 \subseteq \hat{S}$ is the set of **initial states** ($\hat{S}$ is the potential state space)
  - $\mathcal{E}$ is the set of high-level **events**
  - $\mathcal{N} \subseteq \hat{S} \times \hat{S}$ is the **next-state relation** (function), partitioned by $\mathcal{E}$ such that $\mathcal{N} = \bigcup_{\alpha \in \mathcal{E}} \mathcal{N}_\alpha$
- In Petri nets:
  - Variables are **places** (domain is $\mathbb{N}$)
  - Initial state is **initial marking**
  - Events are **transitions**
  - The next-state function is defined by **weighted (inhibitor) arcs**



Decision Diagrams

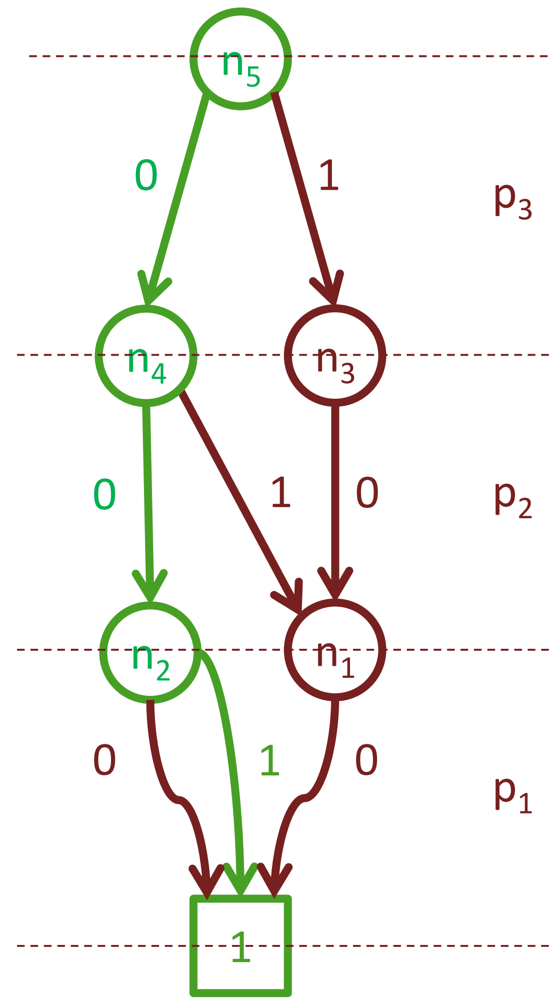
- Encoding sets: Quasi-reduced Ordered **Multi-valued Decision Diagrams (MDD)**
  - **Nodes** encode decisions (evaluation of a variable)
  - **Arcs** encode outcomes (values of a variable)
  - **Terminal nodes** encode result (**0** or **1**)
    - Arcs leading to **0** are not drawn
  - Ordered: same **variable order** on all paths
  - Quasi-reduced: no 2 nodes with the same children
- **Semantics:**
  - Each path from the root node to **1** encodes a tuple
  - Components assume the values written on the arcs



$$S(n_5) = \{(0,0,0), (1,0,0), (0,1,0), (0,0,1)\}$$

Decision Diagrams

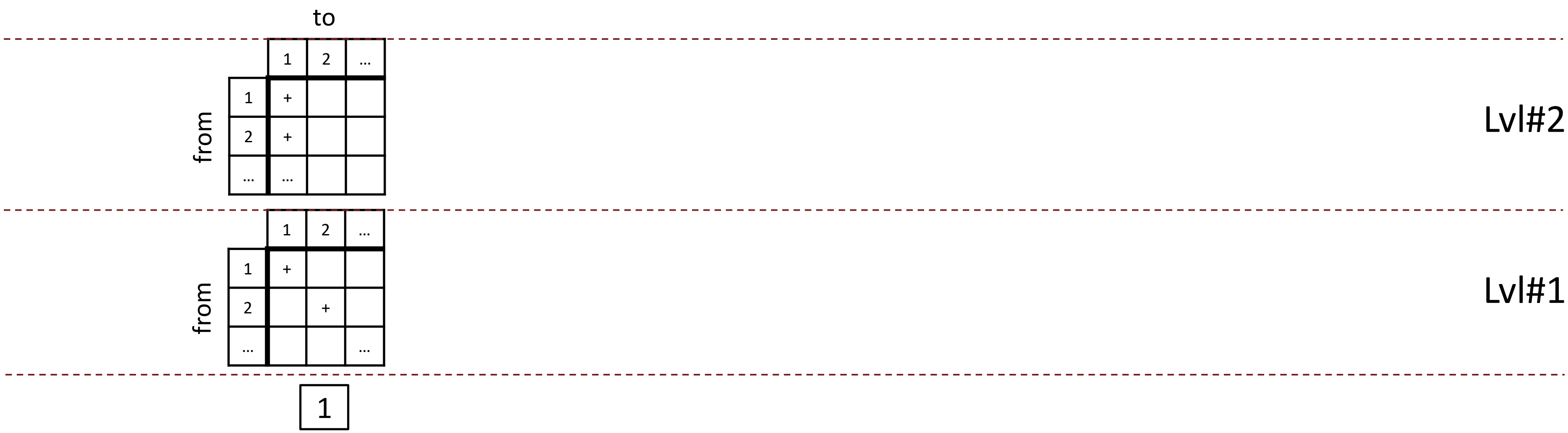
- Encoding sets: Quasi-reduced Ordered **Multi-valued Decision Diagrams (MDD)**
  - **Nodes** encode decisions (evaluation of a variable)
  - **Arcs** encode outcomes (values of a variable)
  - **Terminal nodes** encode result (**0** or **1**)
    - Arcs leading to **0** are not drawn
  - Ordered: same **variable order** on all paths
  - Quasi-reduced: no 2 nodes with the same children
- **Semantics:**
  - Each path from the root node to **1** encodes a tuple
  - Components assume the values written on the arcs
- Efficient recursive operations
  - Heavy **caching**



$S(n_5) = \{(0,0,0), (1,0,0), (0,1,0), (0,0,1)\}$

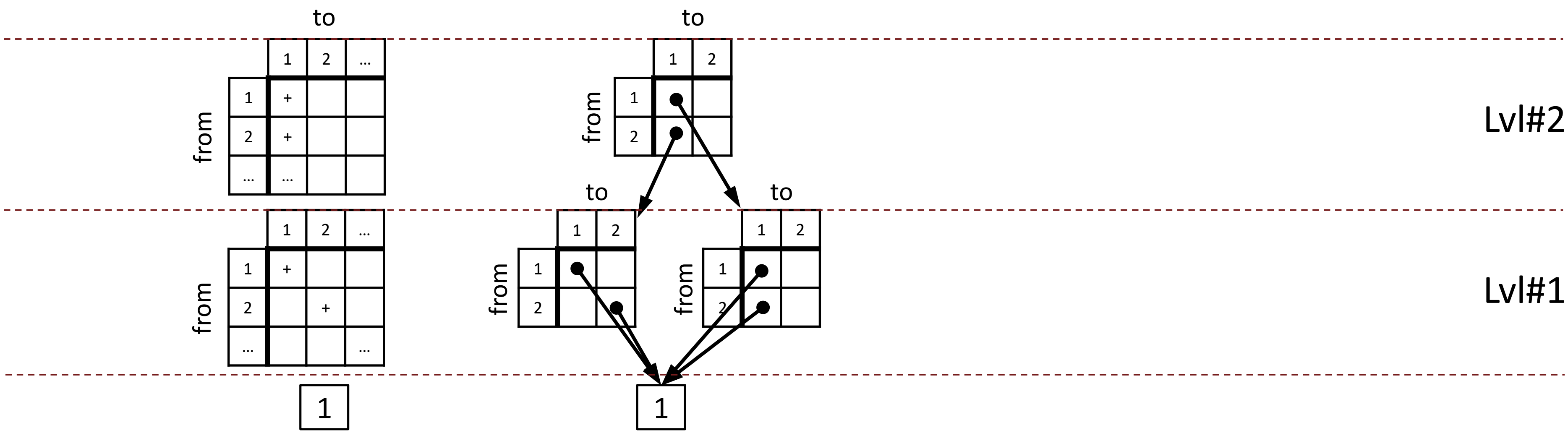
Abstract Next-State Descriptor (Marussy et al, PN'17)

- General description of MDD-like next-state representations
- An **Abstract Next-State Descriptor (ANSD)** is a tuple $(\mathcal{D}, lvl, next)$:
  - $\mathcal{D}$ is the set of **descriptors** (≈MDD nodes) incl. terminal **empty** and **identity**;
  - $lvl : \mathcal{D} \to \mathbb{N}$ is the **level** function (used to assign descriptors to variables);
  - $next : \mathcal{D} \times \mathbb{N} \times \mathbb{N} \to \mathcal{D}$ is the **indexing** function that computes a child descriptor one level lower from a (*source*, *target*) index pair
- Can be regarded as a **common interface** for…
  - Kroenecker matrices



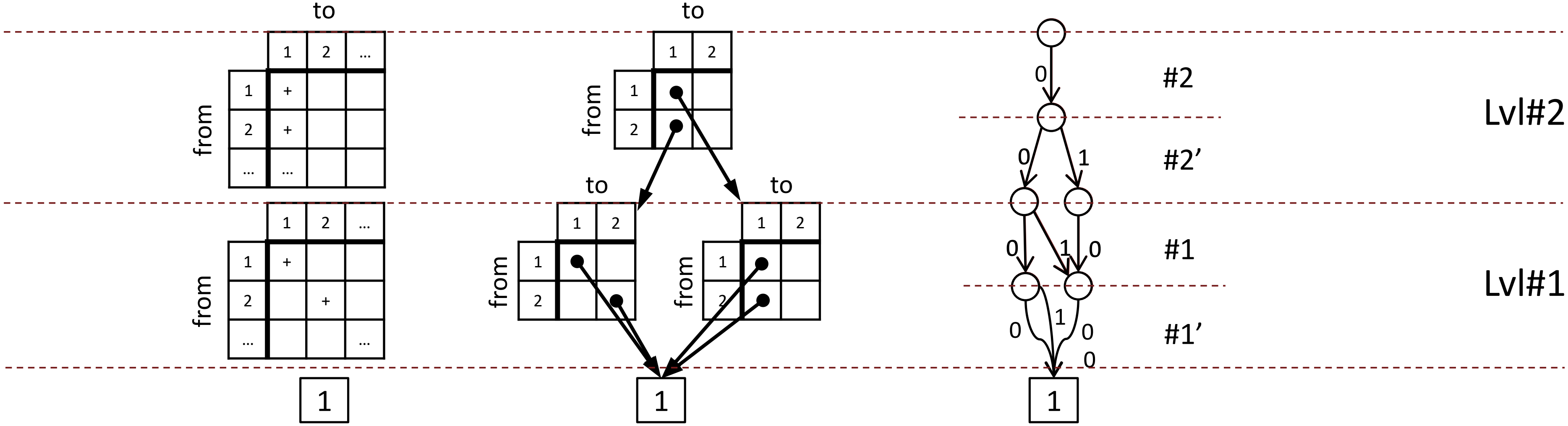
Abstract Next-State Descriptor (Marussy et al, PN'17)

- General description of MDD-like next-state representations
- An **Abstract Next-State Descriptor (ANSD)** is a tuple $(\mathcal{D}, lvl, next)$:
  - $\mathcal{D}$ is the set of **descriptors** (≈MDD nodes) incl. terminal **empty** and **identity**;
  - $lvl : \mathcal{D} \to \mathbb{N}$ is the **level** function (used to assign descriptors to variables);
  - $next : \mathcal{D} \times \mathbb{N} \times \mathbb{N} \to \mathcal{D}$ is the **indexing** function that computes a child descriptor one level lower from a (*source*, *target*) index pair
- Can be regarded as a **common interface** for…
  - Kroenecker matrices, matrix diagrams



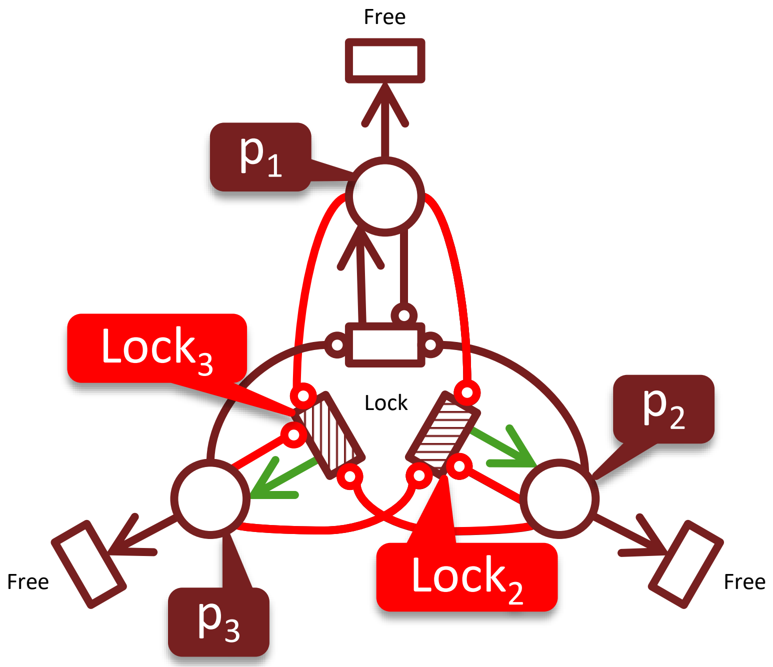
Abstract Next-State Descriptor (Marussy et al, PN'17)

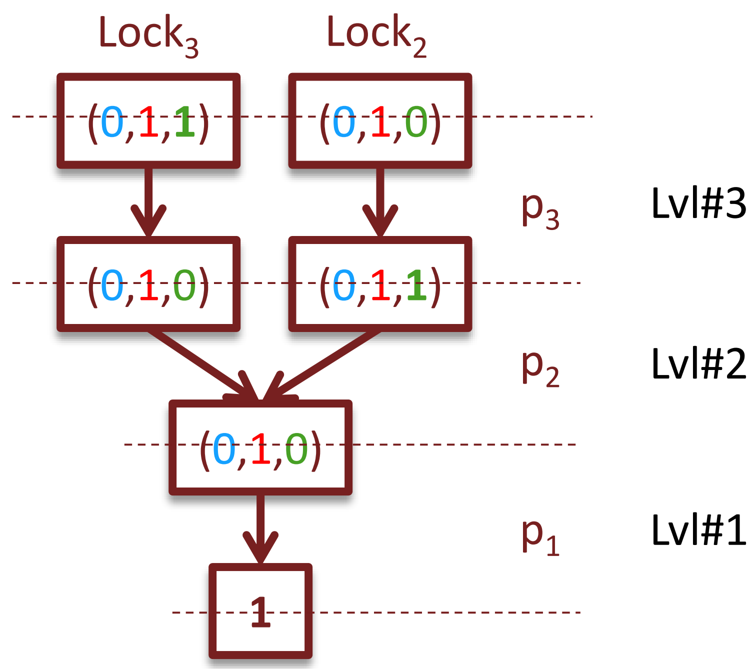
- General description of MDD-like next-state representations
- An **Abstract Next-State Descriptor (ANSD)** is a tuple $(\mathcal{D}, lvl, next)$:
  - $\mathcal{D}$ is the set of **descriptors** (≈MDD nodes) incl. terminal **empty** and **identity**;
  - $lvl : \mathcal{D} \to \mathbb{N}$ is the **level** function (used to assign descriptors to variables);
  - $next : \mathcal{D} \times \mathbb{N} \times \mathbb{N} \to \mathcal{D}$ is the **indexing** function that computes a child descriptor one level lower from a (*source*, *target*) index pair
- Can be regarded as a **common interface** for…
  - Kroenecker matrices, matrix diagrams, MDDs with $2k$ levels, etc.



Abstract Next-State Descriptor (Marussy et al, PN'17)

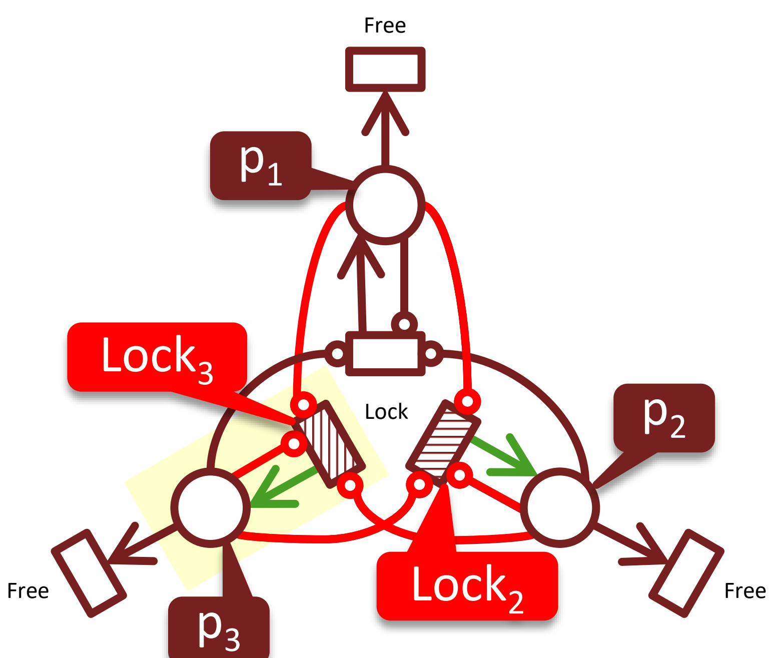
- Very simple representation **for Petri nets**:
  - Descriptors encode transition effect on a single place
  - $d = \langle W^-(t,p), W^\circ(t,p), W^+(t,p), d' \rangle$ (+terminal identity **1**)
  - $next(d,i,j) = \begin{cases} d', & w^- \le i < w^\circ \wedge j = i - w^- + w^+ \\ \mathbf{0}, & \text{otherwise} \end{cases}$

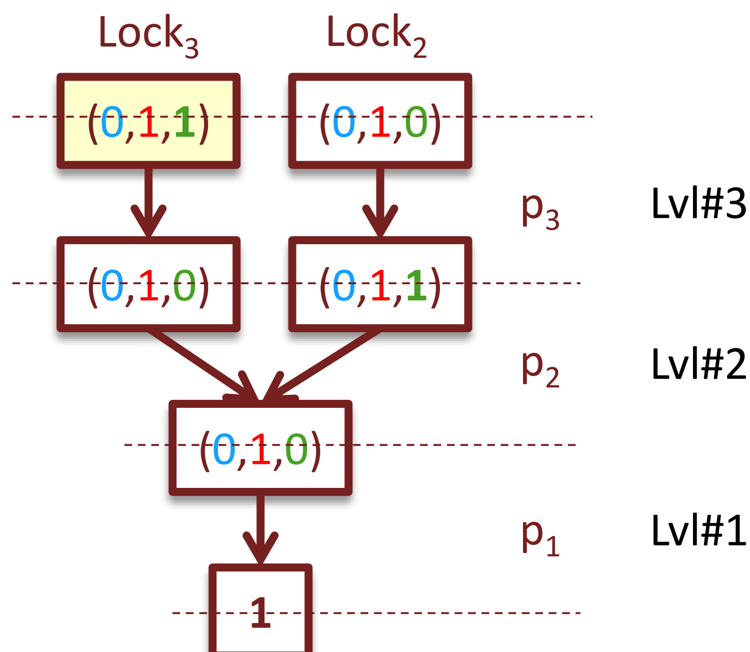




Abstract Next-State Descriptor (Marussy et al, PN'17)

- Very simple representation **for Petri nets**:
  - Descriptors encode transition effect on a single place
  - $d = \langle W^-(t,p), W^\circ(t,p), W^+(t,p), d' \rangle$ (+terminal identity **1**)
  - $next(d,i,j) = \begin{cases} d', & w^- \le i < w^\circ \wedge j = i - w^- + w^+ \\ \mathbf{0}, & \text{otherwise} \end{cases}$

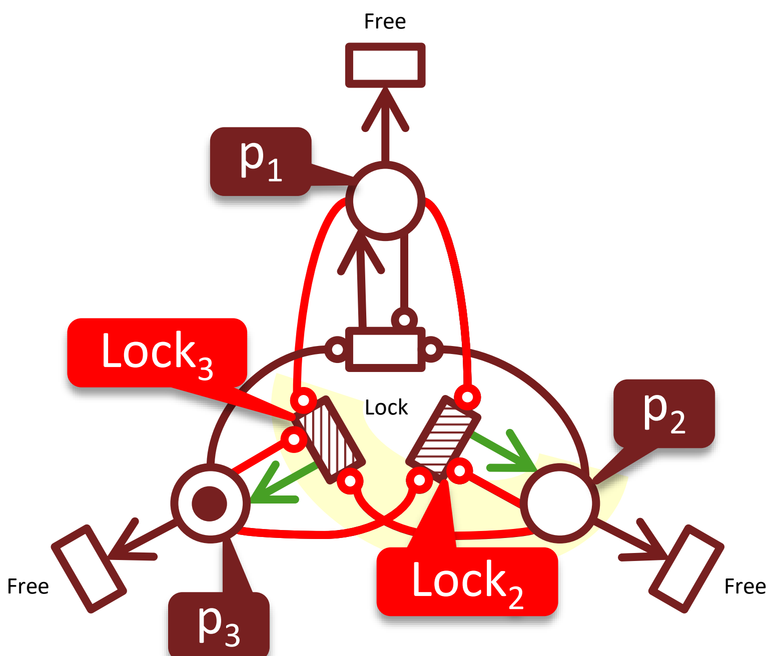


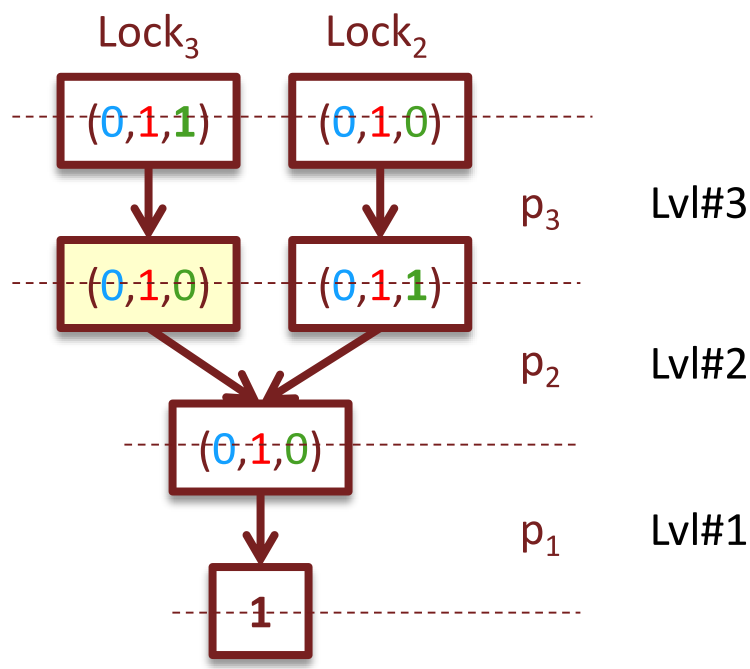


V. Molnár, I. Majzik: *Saturation Enhanced with Conditional Locality: Application to Petri Nets*, PN'19, June 27.

Abstract Next-State Descriptor (Marussy et al, PN'17)

- Very simple representation **for Petri nets**:
  - Descriptors encode transition effect on a single place
  - $d = \langle W^{-}(t,p), W^{\circ}(t,p), W^{+}(t,p), d' \rangle$ (+terminal identity **1**)
  - $next(d,i,j) = \begin{cases} d', & w^{-} \leq i < w^{\circ} \wedge j = i - w^{-} + w^{+} \\ \mathbf{0}, & \text{otherwise} \end{cases}$





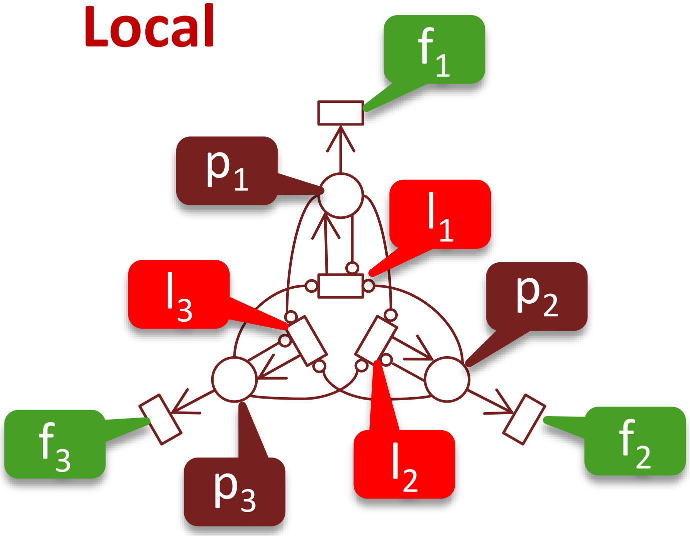
Dependencies & Locality

- Basic **dependencies** between events and variables:

Locally invariant (–)	Locally read-only (r)	Locally read-write (rw)
Value of variable does not affect the outcome of event	Value of variable does not change but affects outcome	Value of variable can be changed by the event

Local

	p_1	p_2	p_3
l_1	rw	r	r
l_2	r	rw	r
l_3	r	r	rw
f_1	rw	–	–
f_2	–	rw	–
f_3	–	–	rw



- **Locality:** Events usually depend on a subset of variables only
  - Next-state relation can be defined over these **supporting** variables

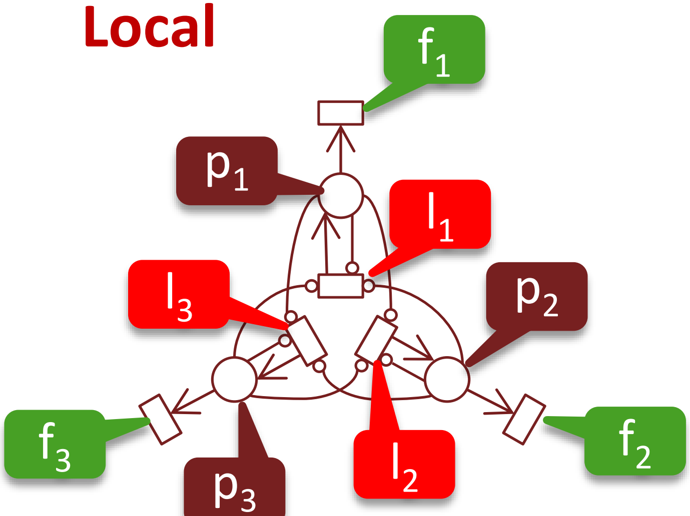
Dependencies & Locality

- Basic **dependencies** between events and variables:

Locally invariant (–)	Locally read-only (r)	Locally read-write (rw)
Value of variable does not affect the outcome of event	Value of variable does not change but affects outcome	Value of variable can be changed by the event

Local

	p_1	p_2	p_3
l_1	rw	r	r
l_2	r	rw	r
l_3	r	r	rw
f_1	rw	–	–
f_2	–	rw	–
f_3	–	–	rw



- **Locality:** Events usually depend on a subset of variables only
  - Next-state relation can be defined over these **supporting** variables

Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable

	p_1	p_2	p_3
l_1	rw	r	r
l_2	r	rw	r
l_3	r	r	rw
f_1	rw	–	–
f_2	–	rw	–
f_3	–	–	rw

Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\left(\mathcal{N}(S^0)\right) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD

	p_1	p_2	p_3
l_1	rw	r	r
l_2	r	rw	r
l_3	r	r	rw
f_1	rw	–	–
f_2	–	rw	–
f_3	–	–	rw

Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD
  - $\mathcal{E}_k = \{\alpha \mid Top(\alpha) = k\}$
  - $\mathcal{N}_k = \bigcup_{\alpha \in \mathcal{E}_k} \mathcal{N}_\alpha$

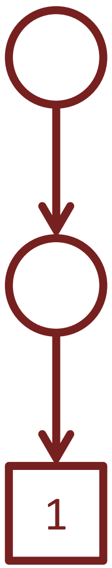
		p_1	p_2	p_3
$\mathcal{E}_3$	l_1	rw	r	r
	l_2	r	rw	r
	l_3	r	r	rw
$\mathcal{E}_1$	f_1	rw	–	–
$\mathcal{E}_2$	f_2	–	rw	–
$\mathcal{E}_3$	f_3	–	–	rw

Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD
  - $\mathcal{E}_k = \{\alpha \mid Top(\alpha) = k\}$
  - $\mathcal{N}_k = \bigcup_{\alpha \in \mathcal{E}_k} \mathcal{N}_\alpha$
- **Saturated MDD node** n:
  - $S(n) = S(n) \cup \mathcal{N}_k(S(n))$ where $k = lvl(n)$
  - Children of n are saturated (**0** and **1** are saturated)
- Starting from a node n encoding $S(n) = S^0$
  - **Recursively saturate** children

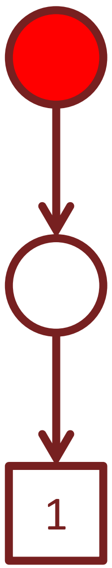
Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD
  - $\mathcal{E}_k = \{\alpha \mid Top(\alpha) = k\}$
  - $\mathcal{N}_k = \bigcup_{\alpha \in \mathcal{E}_k} \mathcal{N}_\alpha$
- **Saturated MDD node** n:
  - $S(n) = S(n) \cup \mathcal{N}_k(S(n))$ where $k = lvl(n)$
  - Children of n are saturated (**0** and **1** are saturated)
- Starting from a node n encoding $S(n) = S^0$
  - **Recursively saturate** children then compute fixed point



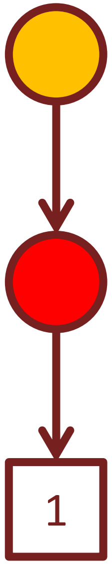
Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD
  - $\mathcal{E}_k = \{\alpha \mid Top(\alpha) = k\}$
  - $\mathcal{N}_k = \bigcup_{\alpha \in \mathcal{E}_k} \mathcal{N}_\alpha$
- **Saturated MDD node** n:
  - $S(n) = S(n) \cup \mathcal{N}_k(S(n))$ where $k = lvl(n)$
  - Children of n are saturated (**0** and **1** are saturated)
- Starting from a node n encoding $S(n) = S^0$
  - **Recursively saturate** children then compute fixed point



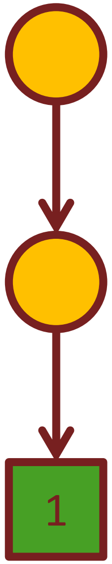
Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD
  - $\mathcal{E}_k = \{\alpha \mid Top(\alpha) = k\}$
  - $\mathcal{N}_k = \bigcup_{\alpha \in \mathcal{E}_k} \mathcal{N}_\alpha$
- **Saturated MDD node** n:
  - $S(n) = S(n) \cup \mathcal{N}_k(S(n))$ where $k = lvl(n)$
  - Children of n are saturated (**0** and **1** are saturated)
- Starting from a node n encoding $S(n) = S^0$
  - **Recursively saturate** children then compute fixed point



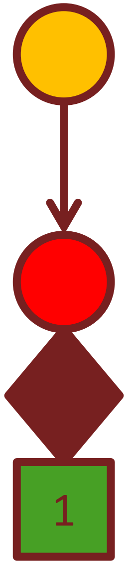
Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD
  - $\mathcal{E}_k = \{\alpha \mid Top(\alpha) = k\}$
  - $\mathcal{N}_k = \bigcup_{\alpha \in \mathcal{E}_k} \mathcal{N}_\alpha$
- **Saturated MDD node** n:
  - $S(n) = S(n) \cup \mathcal{N}_k(S(n))$ where $k = lvl(n)$
  - Children of n are saturated (**0** and **1** are saturated)
- Starting from a node n encoding $S(n) = S^0$
  - **Recursively saturate** children then compute fixed point



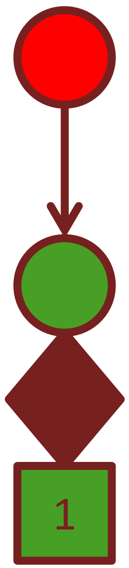
Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD
  - $\mathcal{E}_k = \{\alpha \mid Top(\alpha) = k\}$
  - $\mathcal{N}_k = \bigcup_{\alpha \in \mathcal{E}_k} \mathcal{N}_\alpha$
- **Saturated MDD node** n:
  - $S(n) = S(n) \cup \mathcal{N}_k(S(n))$ where $k = lvl(n)$
  - Children of n are saturated (**0** and **1** are saturated)
- Starting from a node n encoding $S(n) = S^0$
  - **Recursively saturate** children then compute fixed point



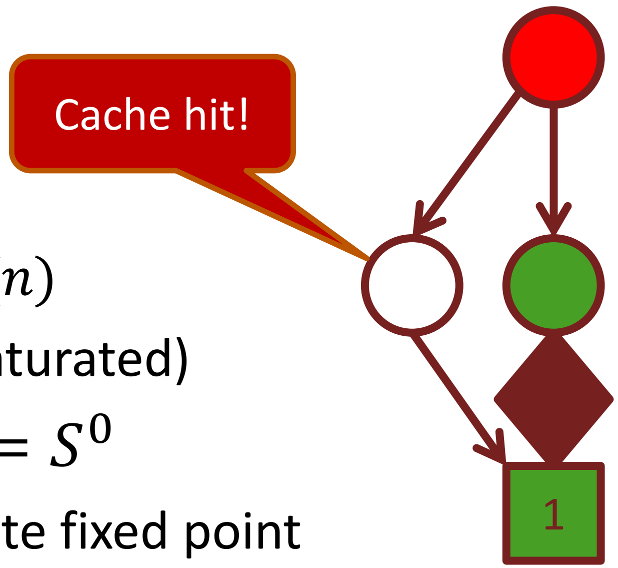
Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD
  - $\mathcal{E}_k = \{\alpha \mid Top(\alpha) = k\}$
  - $\mathcal{N}_k = \bigcup_{\alpha \in \mathcal{E}_k} \mathcal{N}_\alpha$
- **Saturated MDD node** n:
  - $S(n) = S(n) \cup \mathcal{N}_k(S(n))$ where $k = lvl(n)$
  - Children of n are saturated (**0** and **1** are saturated)
- Starting from a node n encoding $S(n) = S^0$
  - **Recursively saturate** children then compute fixed point



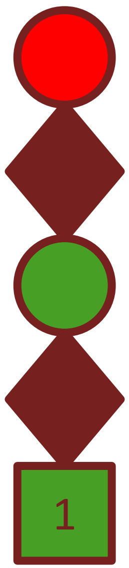
Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD
  - $\mathcal{E}_k = \{\alpha \mid Top(\alpha) = k\}$
  - $\mathcal{N}_k = \bigcup_{\alpha \in \mathcal{E}_k} \mathcal{N}_\alpha$
- **Saturated MDD node** n:
  - $S(n) = S(n) \cup \mathcal{N}_k(S(n))$ where $k = lvl(n)$
  - Children of n are saturated (**0** and **1** are saturated)
- Starting from a node n encoding $S(n) = S^0$
  - **Recursively saturate** children then compute fixed point



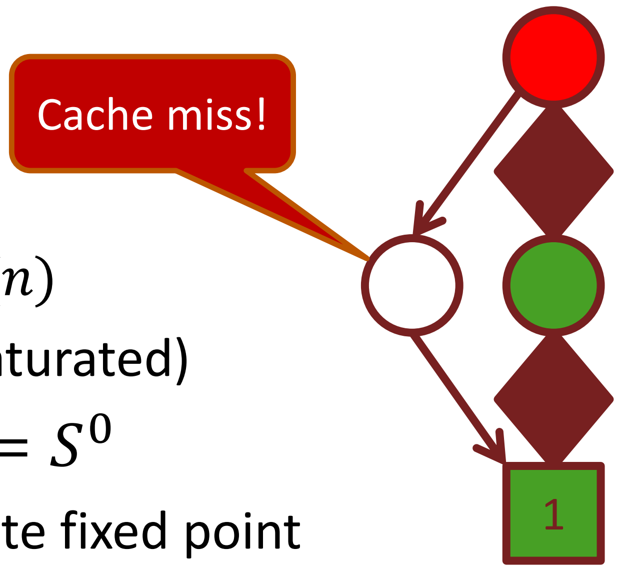
Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD
  - $\mathcal{E}_k = \{\alpha \mid Top(\alpha) = k\}$
  - $\mathcal{N}_k = \bigcup_{\alpha \in \mathcal{E}_k} \mathcal{N}_\alpha$
- **Saturated MDD node** n:
  - $S(n) = S(n) \cup \mathcal{N}_k(S(n))$ where $k = lvl(n)$
  - Children of n are saturated (**0** and **1** are saturated)
- Starting from a node n encoding $S(n) = S^0$
  - **Recursively saturate** children then compute fixed point



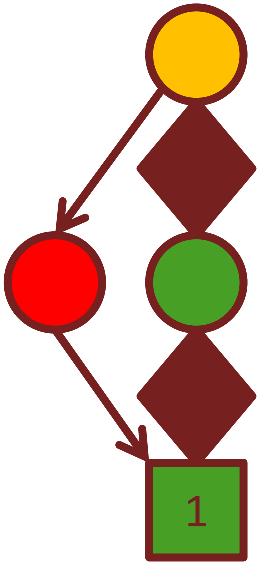
Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD
  - $\mathcal{E}_k = \{\alpha \mid Top(\alpha) = k\}$
  - $\mathcal{N}_k = \bigcup_{\alpha \in \mathcal{E}_k} \mathcal{N}_\alpha$
- **Saturated MDD node** n:
  - $S(n) = S(n) \cup \mathcal{N}_k(S(n))$ where $k = lvl(n)$
  - Children of n are saturated (**0** and **1** are saturated)
- Starting from a node n encoding $S(n) = S^0$
  - **Recursively saturate** children then compute fixed point



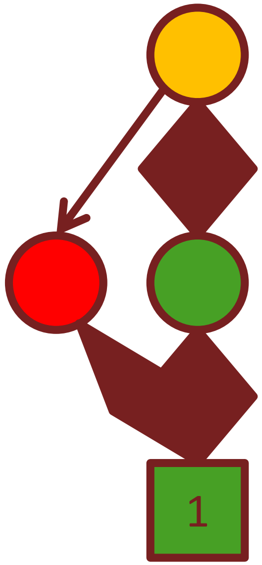
Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD
  - $\mathcal{E}_k = \{\alpha \mid Top(\alpha) = k\}$
  - $\mathcal{N}_k = \bigcup_{\alpha \in \mathcal{E}_k} \mathcal{N}_\alpha$
- **Saturated MDD node** n:
  - $S(n) = S(n) \cup \mathcal{N}_k(S(n))$ where $k = lvl(n)$
  - Children of n are saturated (**0** and **1** are saturated)
- Starting from a node n encoding $S(n) = S^0$
  - **Recursively saturate** children then compute fixed point



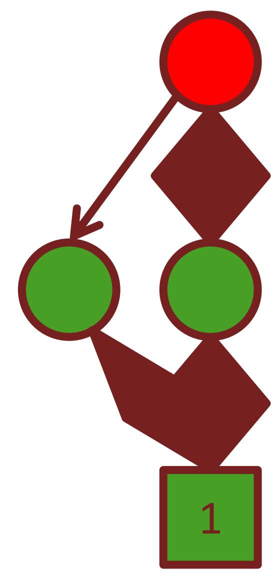
Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD
  - $\mathcal{E}_k = \{\alpha \mid Top(\alpha) = k\}$
  - $\mathcal{N}_k = \bigcup_{\alpha \in \mathcal{E}_k} \mathcal{N}_\alpha$
- **Saturated MDD node** n:
  - $S(n) = S(n) \cup \mathcal{N}_k(S(n))$ where $k = lvl(n)$
  - Children of n are saturated (**0** and **1** are saturated)
- Starting from a node n encoding $S(n) = S^0$
  - **Recursively saturate** children then compute fixed point



Saturation

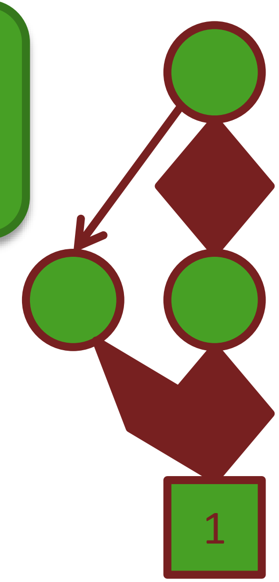
- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD
  - $\mathcal{E}_k = \{\alpha \mid Top(\alpha) = k\}$
  - $\mathcal{N}_k = \bigcup_{\alpha \in \mathcal{E}_k} \mathcal{N}_\alpha$
- **Saturated MDD node n**:
  - $S(n) = S(n) \cup \mathcal{N}_k(S(n))$ where $k = lvl(n)$
  - Children of n are saturated (**0** and **1** are saturated)
- Starting from a node n encoding $S(n) = S^0$
  - **Recursively saturate** children then compute fixed point



Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD
  - $\mathcal{E}_k = \{\alpha \mid Top(\alpha) = k\}$
  - $\mathcal{N}_k = \bigcup_{\alpha \in \mathcal{E}_k} \mathcal{N}_\alpha$
- **Saturated MDD node** n:
  - $S(n) = S(n) \cup \mathcal{N}_k(S(n))$ where $k = lvl(n)$
  - Children of n are saturated (**0** and **1** are saturated)
- Starting from a node n encoding $S(n) = S^0$
  - **Recursively saturate** children then compute fixed point

In the final MDD:
only **saturated** nodes

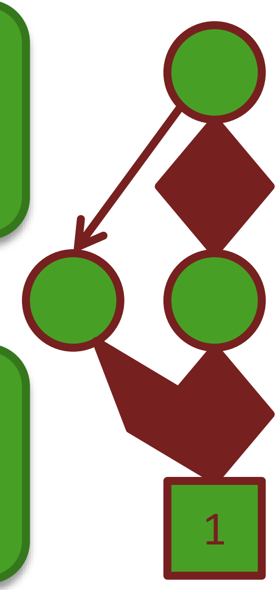


Saturation

- **Saturation** is an algorithm for state space generation of PTSs
- Exploits locality to recursively compute the least fixed point $S = S \cup \mathcal{N}(S)$ such that $S^0 \subseteq S$
  - Equivalent to $S^0 \cup \mathcal{N}(S^0) \cup \mathcal{N}\big(\mathcal{N}(S^0)\big) \cup \cdots = \mathcal{N}^*(S^0)$
- Groups events based on the highest supporting variable
  - $Top(\alpha)$ is the highest supporting variable in the encoding MDD
  - $\mathcal{E}_k = \{\alpha \mid Top(\alpha) = k\}$
  - $\mathcal{N}_k = \bigcup_{\alpha \in \mathcal{E}_k} \mathcal{N}_\alpha$
- **Saturated MDD node** n:
  - $S(n) = S(n) \cup \mathcal{N}_k(S(n))$ where $k = lvl$)
  - Children of n are saturate
- Starting from a node n e
  - **Recursively saturate** child

In the final MDD:
only **saturated** nodes

Less intermediate nodes:
better **performance**



Locally invariant	Locally read-only	Locally read-write
Value of variable does not affect the outcome of event	Value of variable does not change but affects outcome	Value of variable can be changed by the event

Local

Conditionally Local

Enhancing Saturation with Conditional Locality

Conditional Locality

- Additional dependency class between events and variables:

Locally invariant	Locally read-only	Locally read-write
Value of variable does not affect the outcome of event	Value of variable does not change but affects outcome	Value of variable can be changed by the event

Conditionally read-only
For some values of some variables (guard) the event does not change the variable

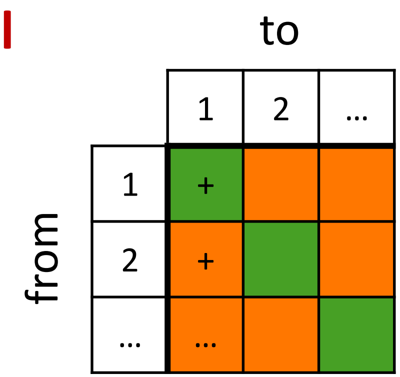
- **Intuition:**
  - Saturation exploits that independent variables **do not change** when firing
  - This might be true for supporting variables as well
    - Definitely true for read-only
    - Conditionally true for conditionally read-only

Conditional locality

- In other words…
  - If an event is conditionally local on a variable, then it can be split s.t. one part is **read-only** and the other is **read-write**

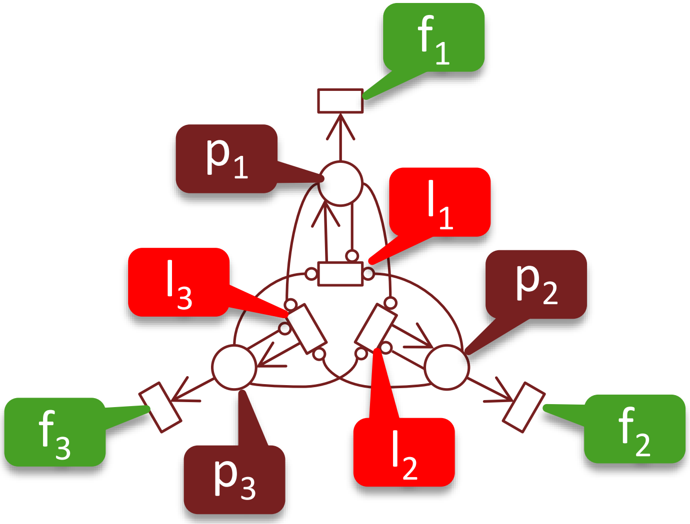
Enhanced Saturation

- Main idea:
  - Compute local fixed point with **conditionally local events**
  - **Remember** and **cache** the effect of higher (unaffected) variables
- Advantages:
  - Conditional Top values **may be lower**
  - Conditionally saturated nodes are **more likely to be final**
- Disadvantages:
  - The effect of higher variables must be remembered
  - (main motivation of constrained saturation)
- With an **ANSD representation** d:
  - $\boldsymbol{next(d,i,i)}$ is the **conditionally local part** (read-only on this level)
    - And the resulting descriptor d' encodes the effect of i on the event!
    - We can fire this part on a lower level any number of times
  - $\boldsymbol{next(d,i,j)}$ with $\boldsymbol{i \neq j}$ is the part we have to **fire on this level**

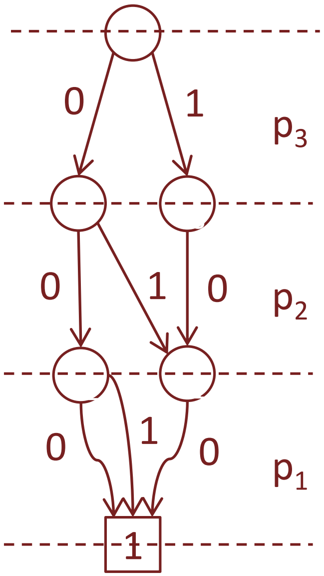


Example

	p_1	p_2	p_3
l_1	rw	r	r
l_2	r	rw	r
l_3	r	r	rw
f_1	rw	–	–
f_2	–	rw	–
f_3	–	–	rw



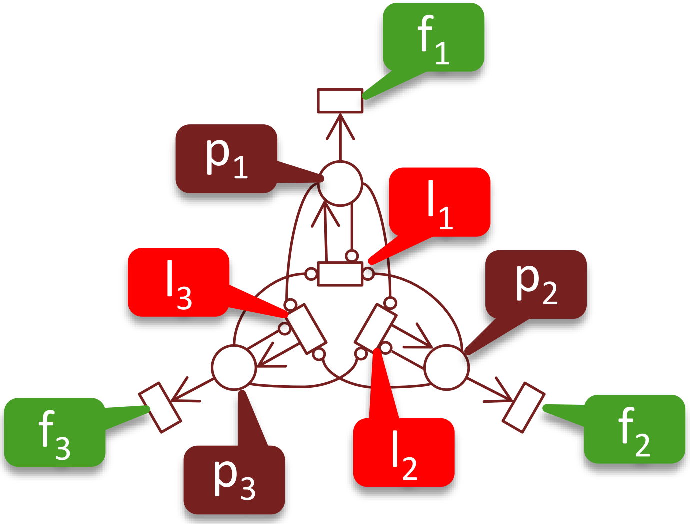
- Assume a variable ordering (p_1, p_2, p_3)



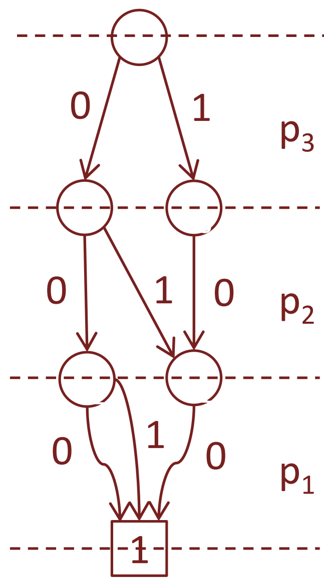
V. Molnár, I. Majzik: *Saturation Enhanced with Conditional Locality: Application to Petri Nets*, PN'19, June 27.

Example

	p_1	p_2	p_3
l_1	rw	r	r
l_2	r	rw	r
l_3	r	r	rw
f_1	rw	–	–
f_2	–	rw	–
f_3	–	–	rw

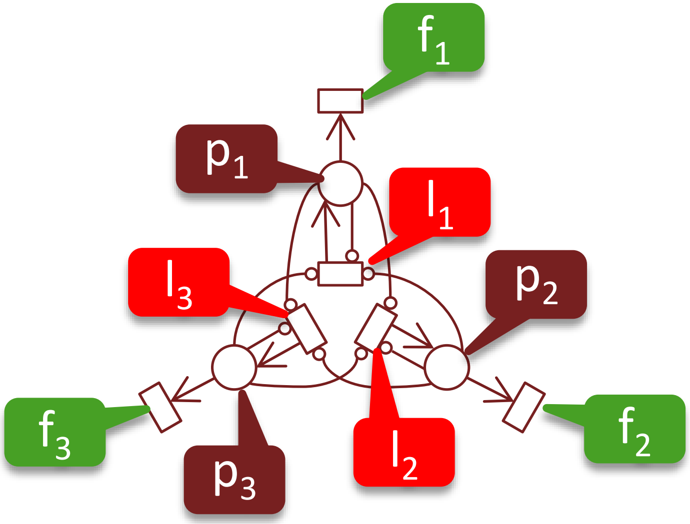


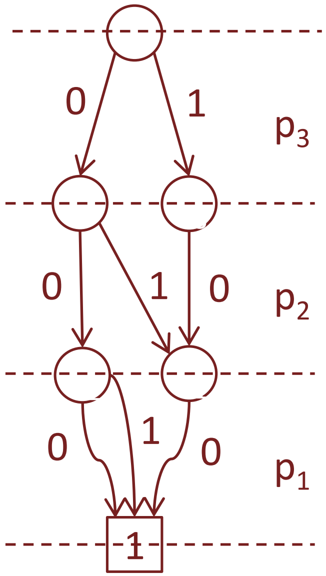
- Assume a variable ordering (p_1, p_2, p_3)
- For **saturation**
  - Everything except f_1 and f_2 must be fired on top level



Example

	p_1	p_2	p_3
l_1	rw	r	r
l_2	r	rw	r
l_3	r	r	rw
f_1	rw	–	–
f_2	–	rw	–
f_3	–	–	rw





- Assume a variable ordering (p_1, p_2, p_3)
- For **saturation**
  - Everything except f_1 and f_2 must be fired on top level
- For saturation with **conditional locality**
  - l_1 and l_2 can also be fired lower if enabled
  - (the fixed point iteration will not change p_2 and/or p_3)

Details and Discussion

- Modified saturation algorithm:
  - $Saturate(n) \rightarrow Saturate(n, \boldsymbol{d})$

    Computes saturated node
    - Saturate now has next-state relation as a parameter (represented by an ANSD)
  - $SatRelProd(n, d) \rightarrow SatRelProd(n, \boldsymbol{d_{sat}}, \boldsymbol{d_{fire}})$

    Computes image of event
    - Relational product still gets next-state relation to fire ($d \rightarrow \boldsymbol{d_{fire}}$)
    - Plus the next-state relation to saturate with ($\boldsymbol{d_{sat}}$, for conditionally local events)
    - Recursion: pass $\boldsymbol{next(d, j, j)}$ for $\boldsymbol{d_{sat}}$ and $\boldsymbol{next(d, i, j)}$ for $\boldsymbol{d_{fire}}$
- A **generalization** of constrained saturation-based methods:
  - Events do not have to be partitioned anymore ($\cancel{\mathcal{E}_k}$), this is **automatic**
  - Contraints/priorities/synchronization can be **directly encoded** in the ANSD
- Overhead?
  - Cache fragmentation because of multiple possible $\boldsymbol{d}$ parameters
  - Offset by **less MDD nodes** created during saturation
  - **Degrades to saturation** without (conditionally) read-only dependencies

Details and Discussion

- Modified saturation algorithm:
  - $Saturate(n) \rightarrow Saturate(n, \mathbf{d})$

    Computes saturated node
    - Saturate now has next-state relation as a parameter (represented by an ANSD)
  - $SatRelProd(n, d) \rightarrow SatRelProd(n, \mathbf{d}_{\mathbf{sat}}, \mathbf{d}_{\mathbf{fire}})$

    Computes image of event
    - Relational product still gets next-state relation to fire ($d \rightarrow \mathbf{d}_{\mathbf{fire}}$)
    - Plus the next-state relation to saturate with ($\mathbf{d}_{\mathbf{sat}}$, for conditionally local events)
    - Recursion: pass $\mathbf{next(d, j, j)}$ for $\mathbf{d}_{\mathbf{sat}}$ and $\mathbf{next(d, i, j)}$ for $\mathbf{d}_{\mathbf{fire}}$
- A **generalization** of constrained saturation-based methods:
  - Events do not have to be partitioned anymore ($\cancel{\mathcal{E}_k}$), this is **automatic**
  - Contraints/priorities/synchronization can be **directly encoded** in the ANSD
- Overhead?

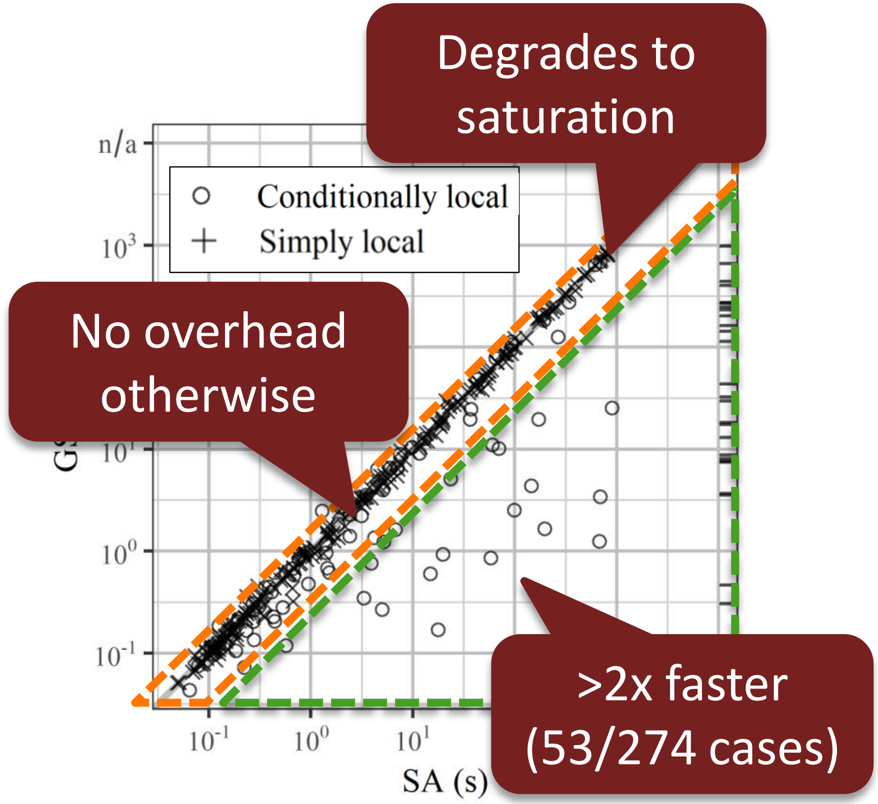
No real overhead expected for Petri nets

Token count may either enable or disable transition, value is not used elsewhere (only two possible child descriptors: d' and $\mathbf{0}$)

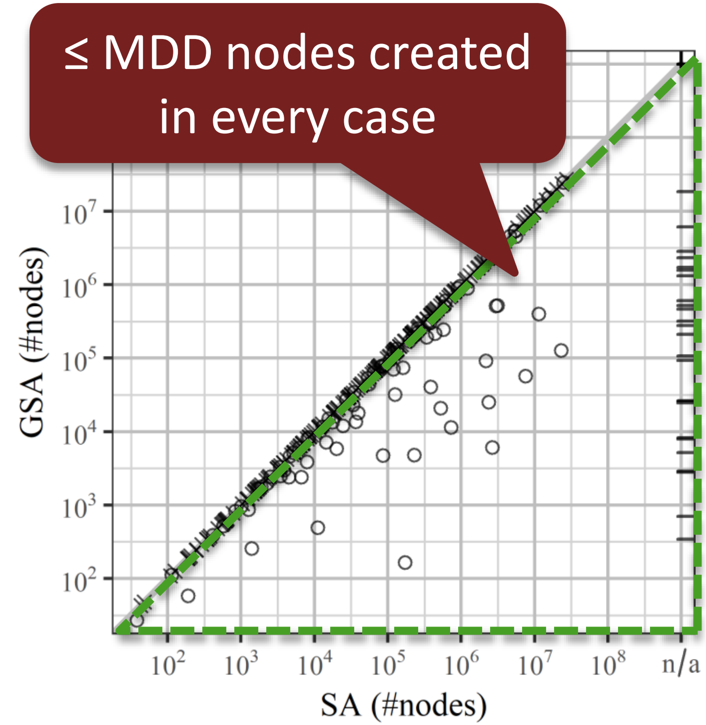
Empirical Evaluation

- Implemented saturation (SA) and generalized saturation (GSA)
- **Models:** (almost) all 743 models from MCC (as of January 2019)
- **Variable orders:**
  - Generated with **sloan** algorithm (recommended by Amparore et al, 2018)

Running time



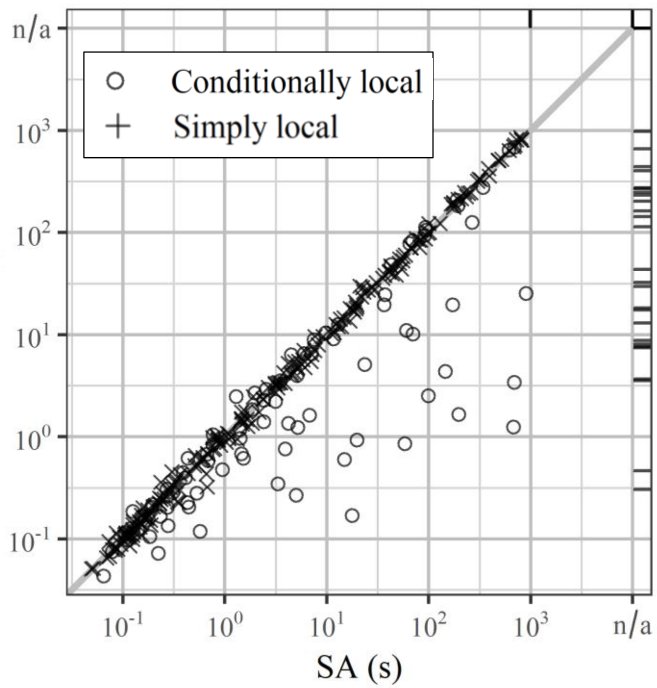
Total # of MDD nodes created



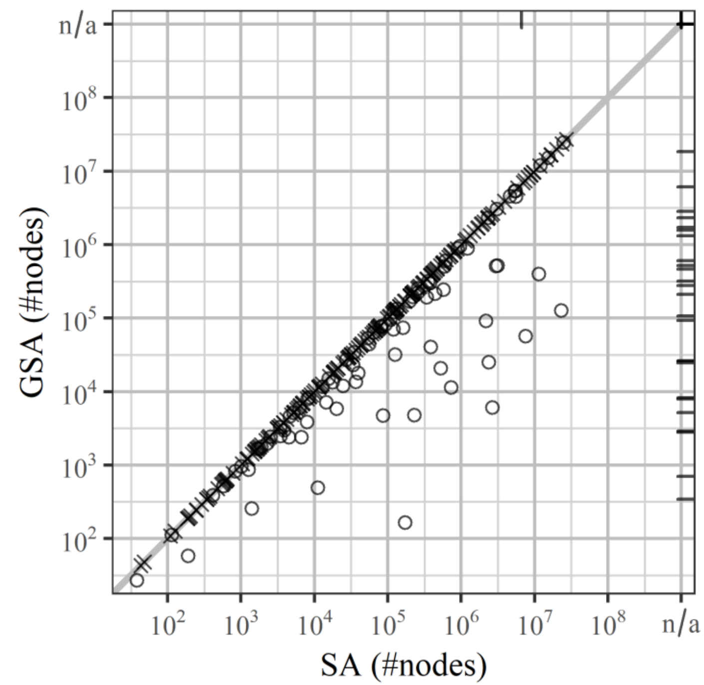
Empirical Evaluation

- Implemented saturation (SA) and generalized saturation (GSA)
- **Models:** (almost) all 743 models from MCC (as of January 2019)
- **Variable orders:**
  - Generated with **sloan** algorithm (recommended by Amparore et al, 2018)

Running time



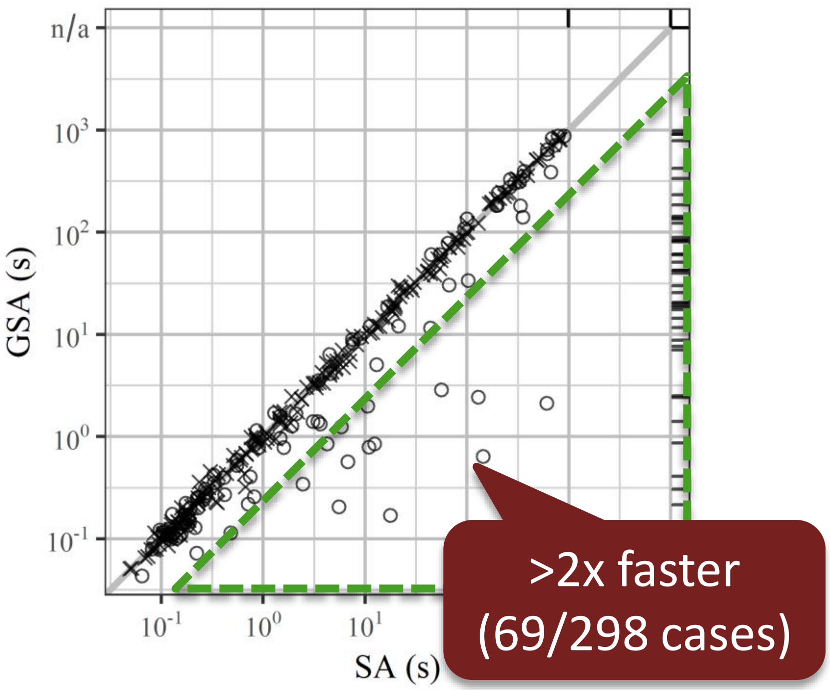
Total # of MDD nodes created



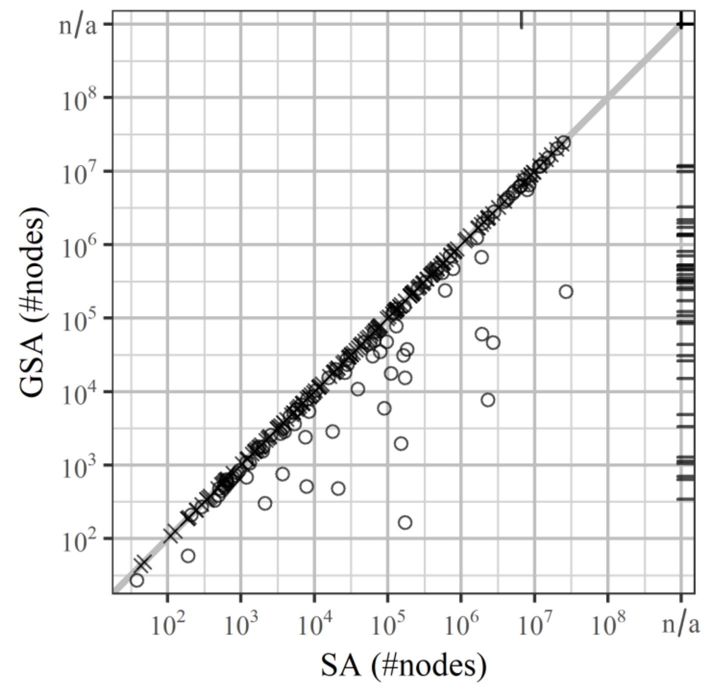
Empirical Evaluation

- Implemented saturation (SA) and generalized saturation (GSA)
- **Models:** (almost) all 743 models from MCC (as of January 2019)
- **Variable orders:**
  - Generated with **sloan** algorithm (recommended by Amparore et al, 2018)
  - **Modified sloan** leaving out read-only dependencies

Running time



Total # of MDD nodes created



Empirical Evaluation

- Implemented saturation (SA) and generalized saturation (GSA)
- **Models:** (almost) all 743 models from MCC (as of January 2019)
- **Variable orders:**
  - Generated with **sloan** algorithm (recommended by Amparore et al, 2018)
  - **Modified sloan** leaving out read-only dependencies

- **Which** algorithm and which variable order?
  - Modified sloan vs. Sloan (MDD size, difference in 117 models)
    - Modified sloan **smaller MDD**: 69/117 **larger MDD**: 39/117
  - GSA with modified sloan vs. SA with sloan
    - GSA **>2x faster**: 78 models **>2x slower**: 16 models

- More research on (*variable ordering*, *algorithm*) pairs is needed

Summary

Saturation Enhanced with Conditional Locality

- **Conditional Locality**
  - Finer definition of event-variable dependencies
- **Enhanced Saturation**
  - No need to partition next-state relation (done automatically)
  - Generalization of constrained saturation-based approaches
  - Enhanced saturation effect may lead to better performance

Application to Petri Nets

- **Evaluation** on models of the MCC
  - Degrades to saturation without conditional locality
  - Often orders of magnitude faster
  - Virtually no overhead otherwise

Future work: investigate more general models (e.g., statecharts)