Characterization of Reachable Attractors using Petri Net Unfoldings

T. Chatain[1], S. Haar[1], L. Jezequel[1], L. Paulevé[2], S. Schwoon[1]

[1]LSV, ENS Cachan / CNRS / Inria
[2]LRI, Université Paris-Sud / CNRS / Inria

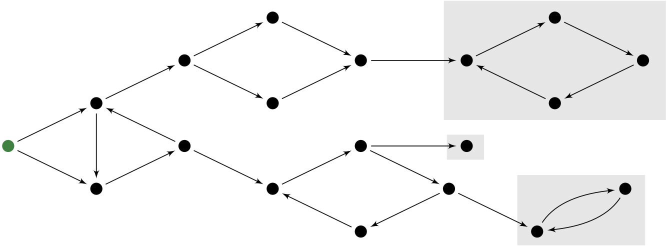
12th Conference on Computational Methods in Systems Biology
17-19 November, 2014 - University of Manchester, UK

Attractors in Discrete Dynamical Models

Models

- Regulatory networks, signalling pathways
- Bio-chemical networks, . . .

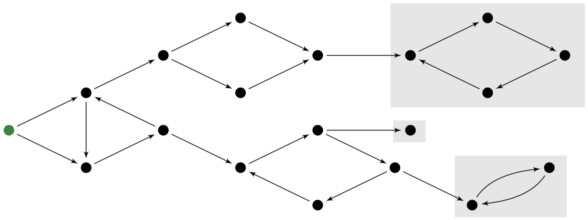
Discrete Dynamics



Attractors

- The "long term" dynamics (limit cycles / fixed points)
- Differentiation processes / homeostasis.

Challenge



Goal: **Exhaustive** list of **attractors reachable from a given state**
(one state of each attractor)

Current approaches

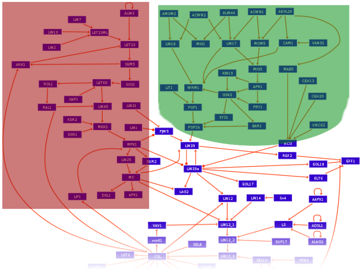
- Explicit/**symbolic** computation of the state graph
- Problem is **inherently hard**: `PSPACE-hard`

Alternatives: approximations using simulations, heuristics w/ topology
(non-exhaustive, no reachability constraint)

Motivations

Exploit **concurrency** in networks

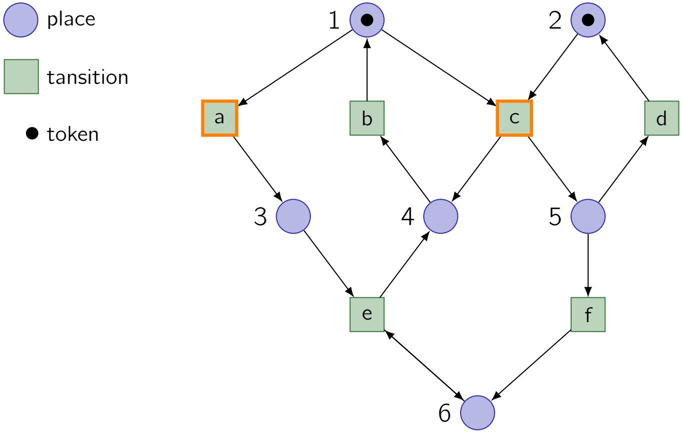
- Many components may evolve independently for a while
- ⇒ generates multiple **spurious interleavings** in the state graph
- Many works in concurrency theory to tackle efficiently such dynamics (partial-order semantics)



Contribution

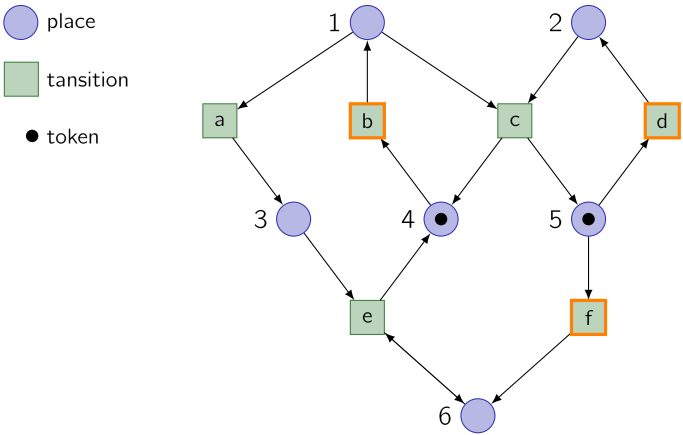
- New algorithm based on **Petri net unfoldings**
- **Complete characterization** of reachable attractors
- Applicable to **any safe Petri net**

Safe Petri Net



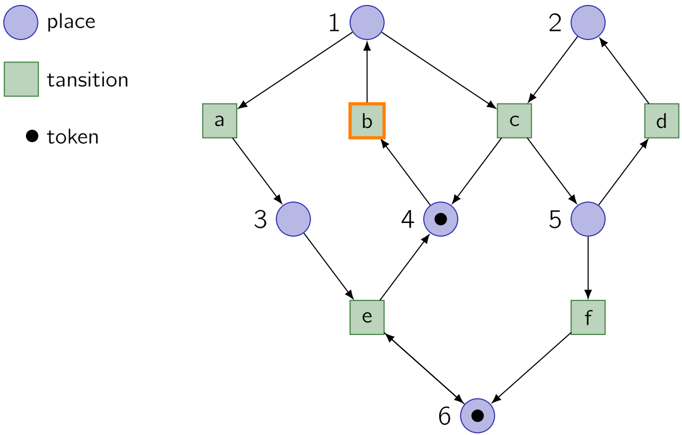
- **Safe** Petri net: at most one token per place
- Marking (state): set of places having one token
- Enabled transition: all (place) parents have one token
- Transition firing (one at a time): 1. empty tokens from parents, 2. add tokens to children

Safe Petri Net



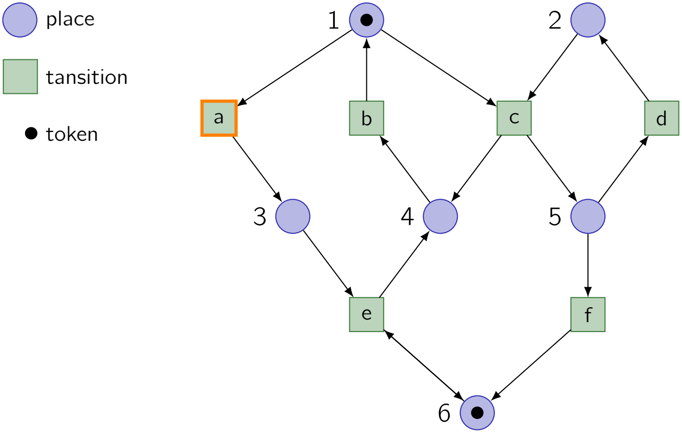
- **Safe** Petri net: at most one token per place
- Marking (state): set of places having one token
- Enabled transition: all (place) parents have one token
- Transition firing (one at a time): 1. empty tokens from parents, 2. add tokens to children

Safe Petri Net



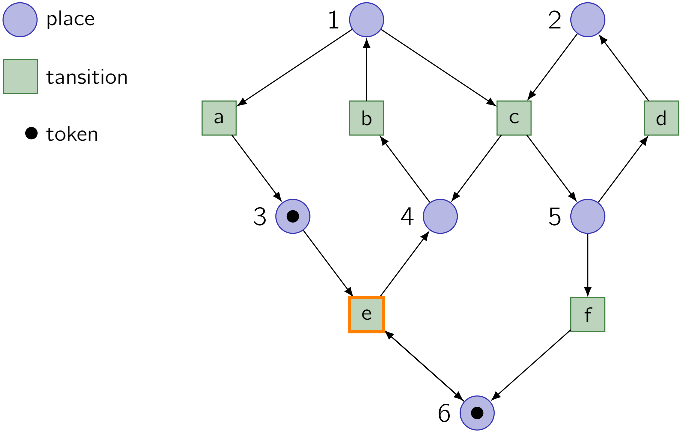
- **Safe** Petri net: at most one token per place
- Marking (state): set of places having one token
- Enabled transition: all (place) parents have one token
- Transition firing (one at a time): 1. empty tokens from parents, 2. add tokens to children

Safe Petri Net



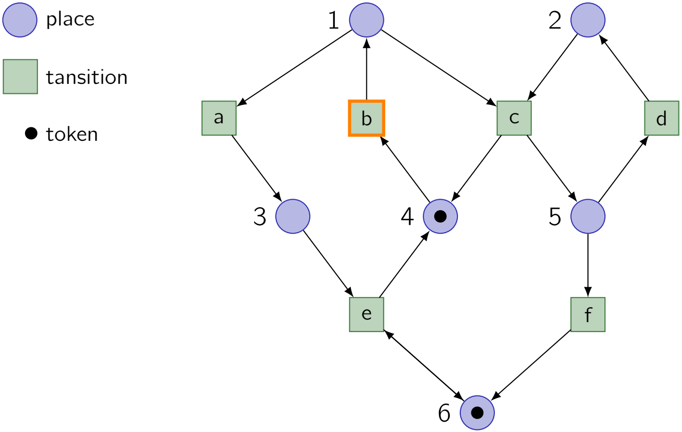
- **Safe** Petri net: at most one token per place
- Marking (state): set of places having one token
- Enabled transition: all (place) parents have one token
- Transition firing (one at a time): 1. empty tokens from parents, 2. add tokens to children

Safe Petri Net



- **Safe** Petri net: at most one token per place
- Marking (state): set of places having one token
- Enabled transition: all (place) parents have one token
- Transition firing (one at a time): 1. empty tokens from parents, 2. add tokens to children

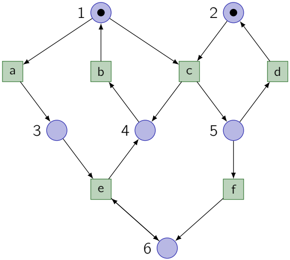
Safe Petri Net



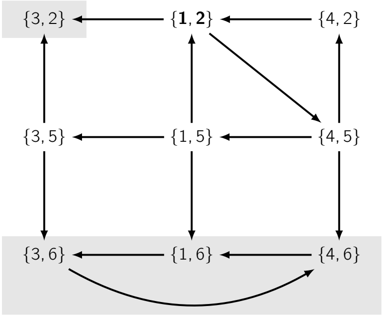
- **Safe** Petri net: at most one token per place
- Marking (state): set of places having one token
- Enabled transition: all (place) parents have one token
- Transition firing (one at a time): 1. empty tokens from parents, 2. add tokens to children

Reachable Attractors in Safe Petri Net

Petri net



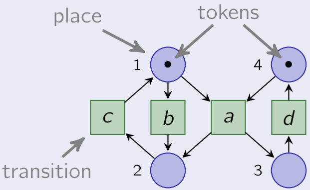
Marking graph



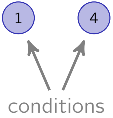
Attractors = Bottom **S**trongly **C**onnected **C**omponents in the marking graph
(from initial marking)

Processes, Branching Processes and Unfoldings

Petri net:

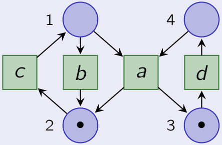


Process: representation of a non-sequential run as a partial order.

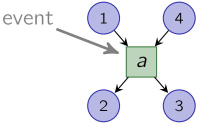


Processes, Branching Processes and Unfoldings

Petri net:

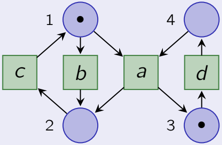


Process: representation of a non-sequential run as a partial order.

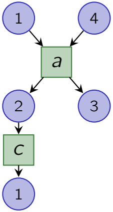


Processes, Branching Processes and Unfoldings

Petri net:

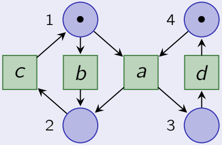


Process: representation of a non-sequential run as a partial order.

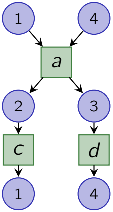


Processes, Branching Processes and Unfoldings

Petri net:

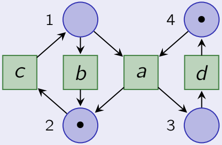


Process: representation of a non-sequential run as a partial order.

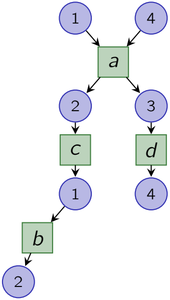


Processes, Branching Processes and Unfoldings

Petri net:

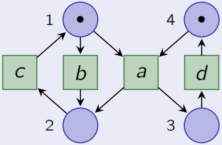


Process: representation of a non-sequential run as a partial order.



Processes, Branching Processes and Unfoldings

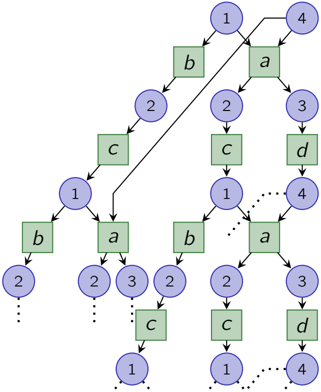
Petri net:



Process: representation of a non-sequential run as a partial order.

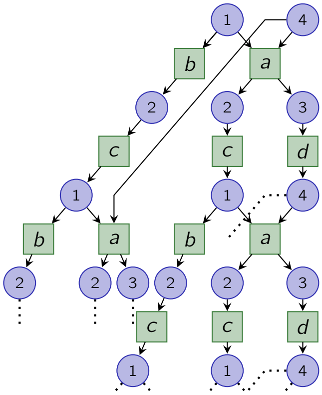
Branching process: representation of several runs.

Unfolding: maximal branching process.



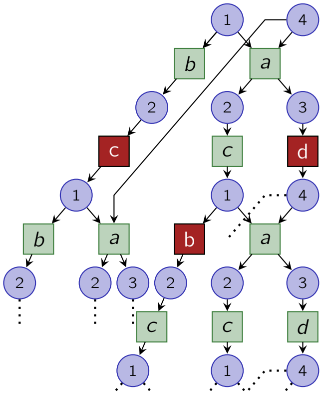
Complete Finite Prefix of Unfolding

- Unfolding is an **infinite acyclic** net



Complete Finite Prefix of Unfolding

- Unfolding is an **infinite acyclic** net



Complete Finite Prefix of Unfolding

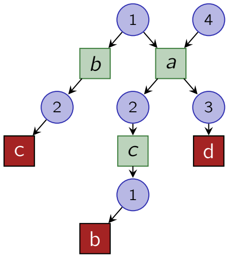
- Unfolding is an **infinite acyclic** net

Complete Finite Prefix

- Prefix of the unfolding that contains **all** reachable markings
- When done with care:

  $$\text{size(prefix)} \leq \text{size(marking\_graph)}$$

- Available tools: `Mole`[1], `Cunf`[2], ..



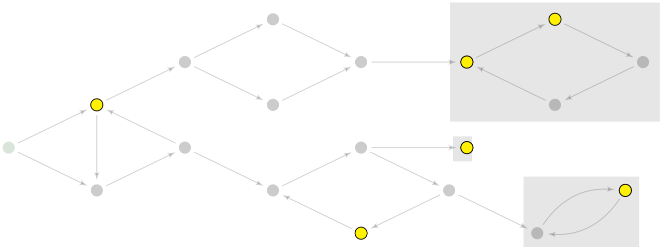
[1]: http://www.lsv.ens-cachan.fr/~schwoon/tools/mole

[2]: https://code.google.com/p/cunf

Identify Attractors with Unfoldings

General idea given a safe Petri net with initial marking m_0

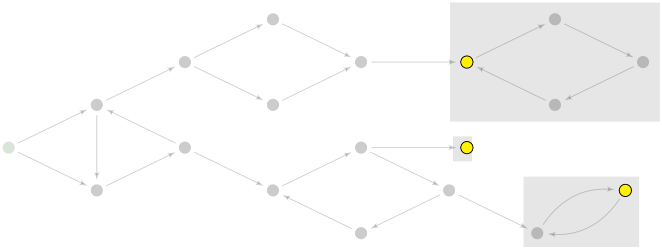
1. Compute complete finite prefix $\mathcal{U}_0$.
2. Extract set of **markings intersecting all attractors**
3. Filter out doubles and false positives



Identify Attractors with Unfoldings

General idea given a safe Petri net with initial marking m_0

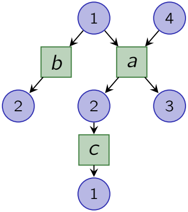
1. Compute complete finite prefix $\mathcal{U}_0$.
2. Extract set of **markings intersecting all attractors**
3. Filter out doubles and false positives



From Complete Prefix to Maximal Configurations

Maximal Configuration:
Marking led by a **maximal process** in the complete finite prefix

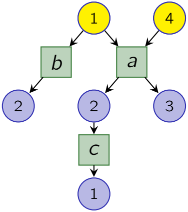
- Can be encoded as SAT.
- All attractors have at least one marking as max. conf.



From Complete Prefix to Maximal Configurations

Maximal Configuration:
Marking led by a **maximal process** in the complete finite prefix

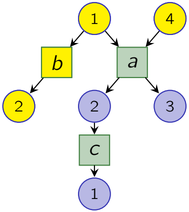
- Can be encoded as SAT.
- All attractors have at least one marking as max. conf.



From Complete Prefix to Maximal Configurations

Maximal Configuration:
Marking led by a **maximal process** in the complete finite prefix

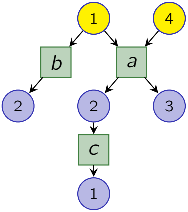
- Can be encoded as SAT.
- All attractors have at least one marking as max. conf.



From Complete Prefix to Maximal Configurations

Maximal Configuration:
Marking led by a **maximal process** in the complete finite prefix

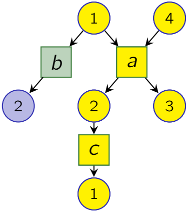
- Can be encoded as SAT.
- All attractors have at least one marking as max. conf.



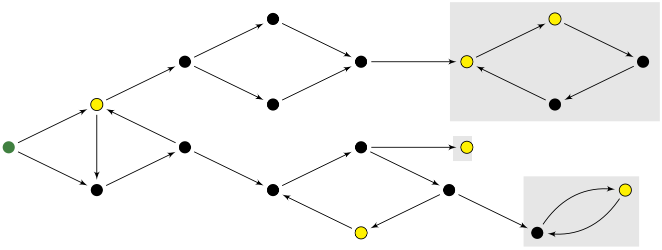
From Complete Prefix to Maximal Configurations

Maximal Configuration:
Marking led by a **maximal process**
in the complete finite prefix

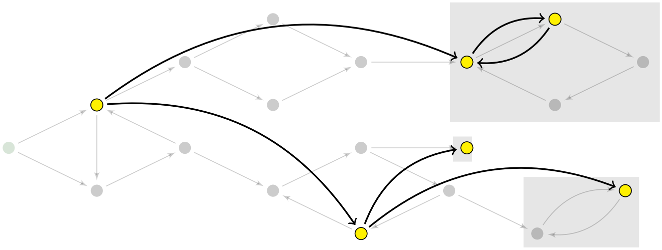
- Can be encoded as SAT.
- All attractors have at least one marking as max. conf.



Two max. conf.: $\{2, 4\}$, $\{1, 3\}$.



From Maximal Configurations to Attractors



Input: safe Petri net with initial marking m_0

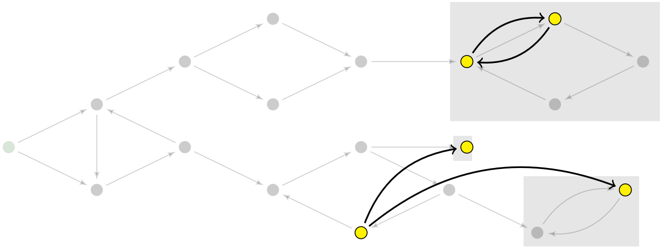
1. Compute complete finite prefix $\mathcal{U}_0$.
2. Compute the maximal configurations $\mathcal{M} = \{m_1, \cdots, m_k\}$
3. For i from 1 to k:
    1. if any marking in $\mathcal{M}$ is reachable from m_i (e.g., using $\mathcal{U}_i$): remove m_i from $\mathcal{M}$.

Output $\mathcal{M}$

Result:

- **Each attractor reachable from** m_0 has **one and only one** marking in $\mathcal{M}$
- Each marking in $\mathcal{M}$ belongs to an attractor reachable from m_0.

From Maximal Configurations to Attractors



Input: safe Petri net with initial marking m_0

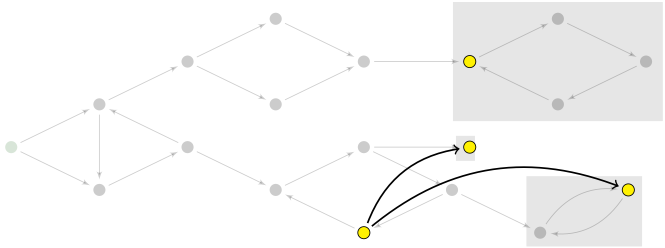
1. Compute complete finite prefix $\mathcal{U}_0$.
2. Compute the maximal configurations $\mathcal{M} = \{m_1, \cdots, m_k\}$
3. For i from 1 to k:
   1. if any marking in $\mathcal{M}$ is reachable from m_i (e.g., using $\mathcal{U}_i$): remove m_i from $\mathcal{M}$.

Output $\mathcal{M}$

Result:

- **Each attractor reachable from** m_0 has **one and only one** marking in $\mathcal{M}$
- Each marking in $\mathcal{M}$ belongs to an attractor reachable from m_0.

From Maximal Configurations to Attractors



Input: safe Petri net with initial marking m_0

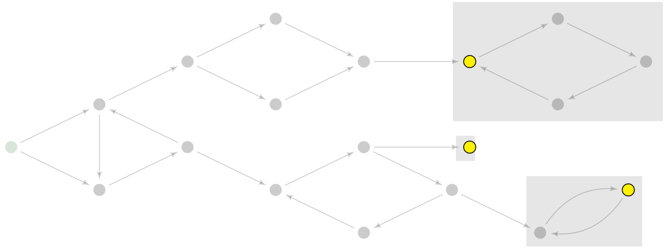
1. Compute complete finite prefix $\mathcal{U}_0$.
2. Compute the maximal configurations $\mathcal{M} = \{m_1, \cdots, m_k\}$
3. For i from 1 to k:
    1. if any marking in $\mathcal{M}$ is reachable from m_i (e.g., using $\mathcal{U}_i$): remove m_i from $\mathcal{M}$.

Output $\mathcal{M}$

Result:

- **Each attractor reachable from** m_0 has **one and only one** marking in $\mathcal{M}$
- Each marking in $\mathcal{M}$ belongs to an attractor reachable from m_0.

From Maximal Configurations to Attractors



Input: safe Petri net with initial marking m_0

1. Compute complete finite prefix $\mathcal{U}_0$.
2. Compute the maximal configurations $\mathcal{M} = \{m_1, \cdots, m_k\}$
3. For i from 1 to k:
    1. if any marking in $\mathcal{M}$ is reachable from m_i (e.g., using $\mathcal{U}_i$): remove m_i from $\mathcal{M}$.

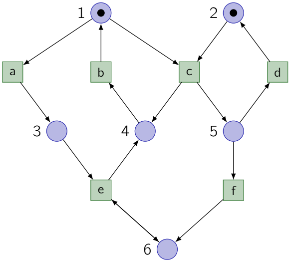
Output $\mathcal{M}$

Result:

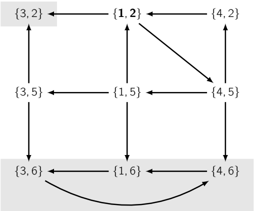
- **Each attractor reachable from** m_0 has **one and only one** marking in $\mathcal{M}$
- Each marking in $\mathcal{M}$ belongs to an attractor reachable from m_0.

Small Example

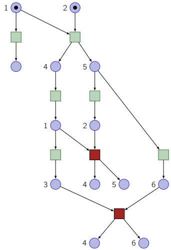
0. Petri net



Marking graph

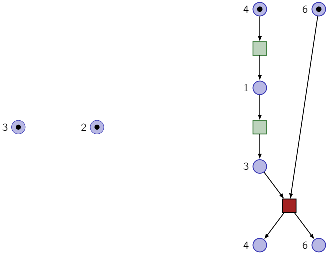


1. Complete Finite Prefix



2. Max. config.: {2, 3}, {4, 5}, {4, 6}

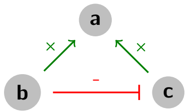
3. Attractors



Safe Petri Net Encoding of Discrete Networks

Discrete Networks

- Set of variables with value in $\mathbb{D} = \{0, \dots, l\}$ (Boolean or multi-valued)
- For each variable i, $f^i : \mathbb{D}^n \to \mathbb{D}$ (typically depends on a few other variables)



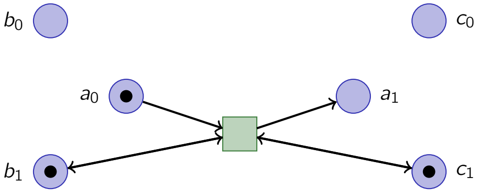
$$f^a(x) = x[b] \wedge x[c]$$
$$f^b(x) = 1$$
$$f^c(x) = \neg x[b]$$

Encoding of asynchronous dynamics using safe Petri net

- One place per variable value
- Transitions for value changes

$f^a(x) = x[b] \wedge x[c]$

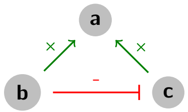
- $a_0 \to a_1$ when $b_1 \wedge c_1$
- $a_1 \to a_0$ when b_0
- $a_1 \to a_0$ when c_0



Safe Petri Net Encoding of Discrete Networks

Discrete Networks

- Set of variables with value in $\mathbb{D} = \{0, \ldots, l\}$ (Boolean or multi-valued)
- For each variable i, $f^i : \mathbb{D}^n \to \mathbb{D}$ (typically depends on a few other variables)



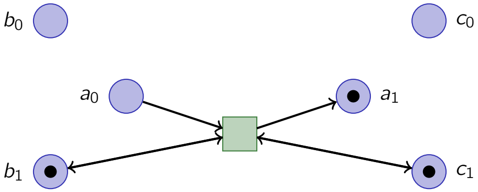
$$f^a(x) = x[b] \land x[c]$$
$$f^b(x) = 1$$
$$f^c(x) = \neg x[b]$$

Encoding of asynchronous dynamics using safe Petri net

- One place per variable value
- Transitions for value changes

$f^a(x) = x[b] \land x[c]$

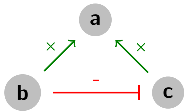
- $a_0 \to a_1$ when $b_1 \land c_1$
- $a_1 \to a_0$ when b_0
- $a_1 \to a_0$ when c_0



Safe Petri Net Encoding of Discrete Networks

Discrete Networks

- Set of variables with value in $\mathbb{D} = \{0, \ldots, l\}$ (Boolean or multi-valued)
- For each variable i, $f^i : \mathbb{D}^n \to \mathbb{D}$ (typically depends on a few other variables)



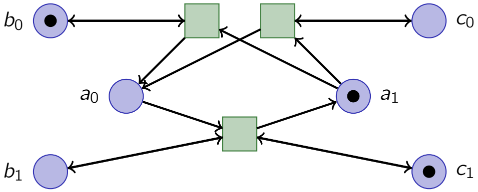
$$f^a(x) = x[b] \wedge x[c]$$
$$f^b(x) = 1$$
$$f^c(x) = \neg x[b]$$

Encoding of asynchronous dynamics using safe Petri net

- One place per variable value
- Transitions for value changes

$f^a(x) = x[b] \wedge x[c]$

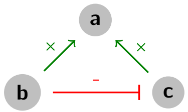
- $a_0 \to a_1$ when $b_1 \wedge c_1$
- $a_1 \to a_0$ when b_0
- $a_1 \to a_0$ when c_0



Safe Petri Net Encoding of Discrete Networks

Discrete Networks

- Set of variables with value in $\mathbb{D} = \{0, \ldots, l\}$ (Boolean or multi-valued)
- For each variable i, $f^i : \mathbb{D}^n \to \mathbb{D}$ (typically depends on a few other variables)



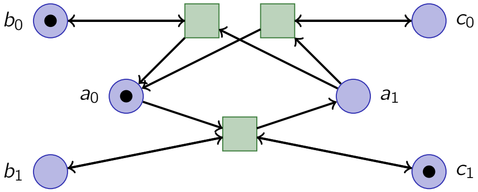
$$f^a(x) = x[b] \land x[c]$$
$$f^b(x) = 1$$
$$f^c(x) = \neg x[b]$$

Encoding of asynchronous dynamics using safe Petri net

- One place per variable value
- Transitions for value changes

$f^a(x) = x[b] \land x[c]$

- $a_0 \to a_1$ when $b_1 \land c_1$
- $a_1 \to a_0$ when b_0
- $a_1 \to a_0$ when c_0



Experiments

Preliminary implementation

Model	nodes	reachable	prefix	max. conf.	attractors	time
Lambda switch	4	46	45	15	2	<1s
Cell cycle	10	112	111	34	1	<1s
ERBB	20	2,963	1,113	302	2	5s
VPC C. elegans	88	152,320	973	1,240	1	15min

Remarks:

- In those examples, most of the time is spent in filtering max. conf.
- There may be many useless max. conf.
- Computing complete finite prefix may be not tractable.

Discussion

Goal: **Exhaustive** list of **attractors reachable from a given state** (one state of each attractor)

Proof of concept

- Relies on complete prefix of **unfolding of Petri nets**
- Exploits concurrency between transitions
- **Generic algorithm** for identifying all the reachable attractors

On-going work for scalability

- More **constraints on candidate maximal configurations** (embed co-reachability constraints → QBF)
- Iterative approach using **partial prefixes** of unfoldings.

Further extensions

- Specialize to discrete/Boolean networks
- More efficient unfolding: symbolic, abstractions, ...

Thank you for your attention.