BIOSTEACKER 2014

Exhaustive search of dynamical properties in Process Hitting using Answer Set Programming

Emna BEN ABDALLAH

MeForBio / IRCCyN / École Centrale de Nantes (Nantes, France)
emma.ben-abdallah@eleves.ec-nantes.fr

Context and Aims

MeForBio team:
Algebric modelling to study
complex dynamical biological systems

Context and Aims

MeForBio team:
Algebric modelling to study
complex dynamical biological systems

1) What are the models?

   Biological Regulatory Networks (BRNs): Studying **gene interactions** with mathematical tools;
   Process Hitting (PH): a new developed model.

Context and Aims

MeForBio team:
Algebric modelling to study
complex dynamical biological systems

1) What are the models?

   Biological Regulatory Networks (BRNs): Studying **gene interactions** with mathematical tools;
   Process Hitting (PH): a new developed model.

2) What do I do?

   Predicting the **evolutions** of the network.

Context and Aims

MeForBio team:
Algebric modelling to study
complex dynamical biological systems

1) What are the models?

   Biological Regulatory Networks (BRNs): Studying **gene interactions** with mathematical tools;
   Process Hitting (PH): a new developed model.

2) What do I do?

   Predicting the **evolutions** of the network.

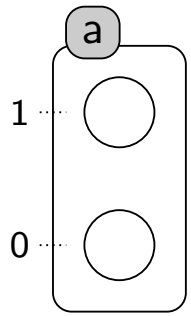
3) What for?

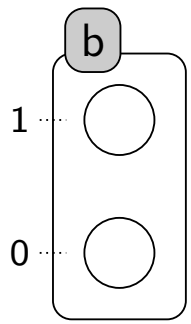
   searching of PH **properties** through ASP (Fixed points, reachability).

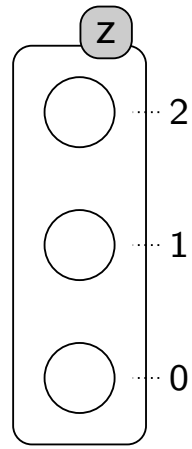
The Process Hitting modeling

Sorts: components a, b, z

The Process Hitting modeling







Sorts: components a, b, z

Processes: local states / levels of expression z_0, z_1, z_2

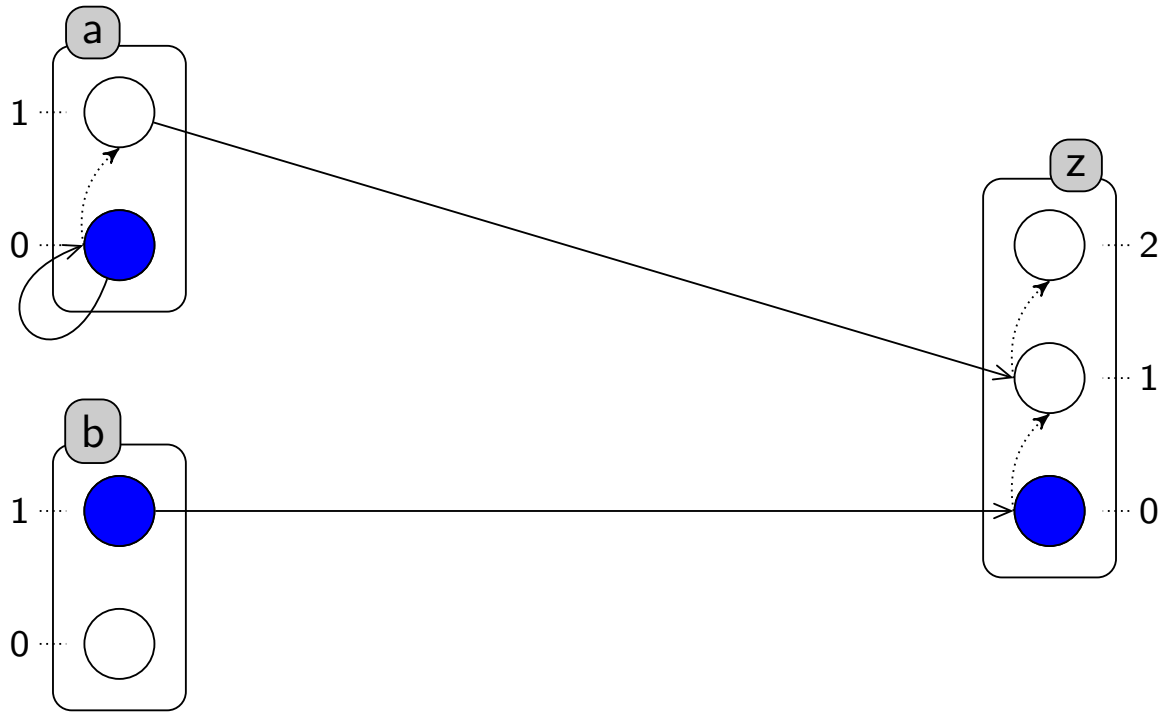
The Process Hitting modeling

Sorts: components $a,\ b,\ z$

Processes: local states / levels of expression $z_0,\ z_1,\ z_2$

States: sets of active processes $\langle a_0, b_1, z_0 \rangle$

The Process Hitting modeling



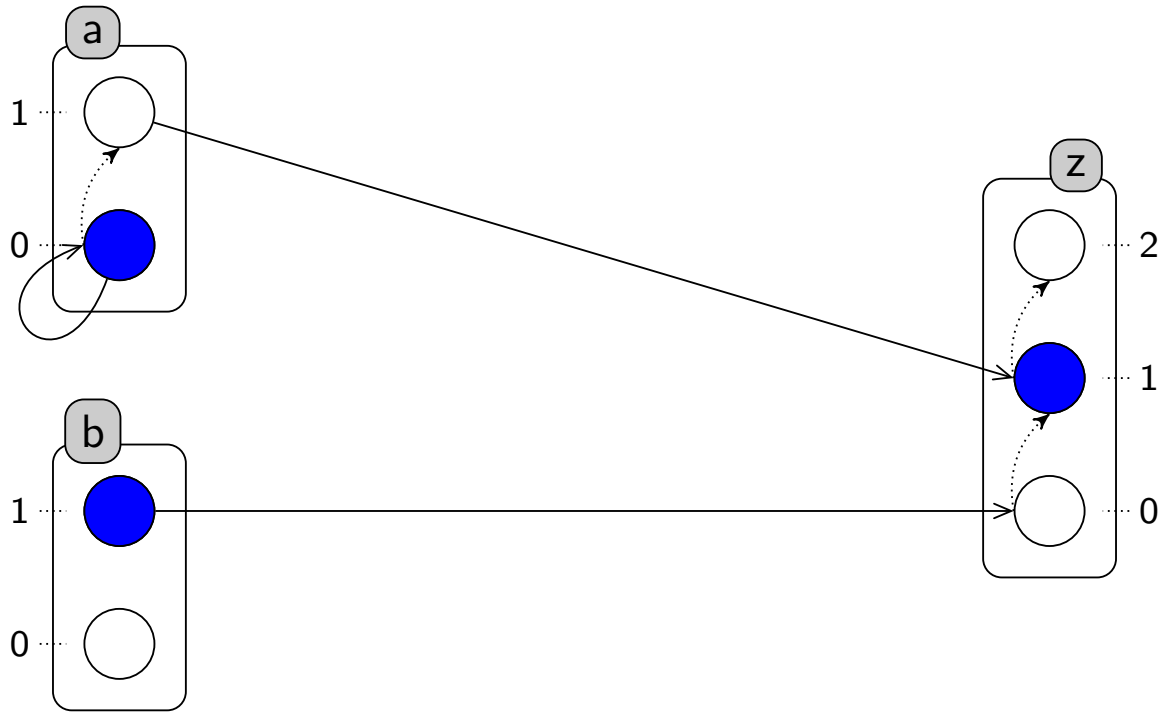
Sorts: components $a,\ b,\ z$

Processes: local states / levels of expression $z_0,\ z_1,\ z_2$

States: sets of active processes $\langle a_0, b_1, z_0 \rangle$

Actions: dynamics $b_1 \longrightarrow z_0 \restriction z_1,\ a_0 \longrightarrow a_0 \restriction a_1,\ a_1 \longrightarrow z_1 \restriction z_2$

The Process Hitting modeling



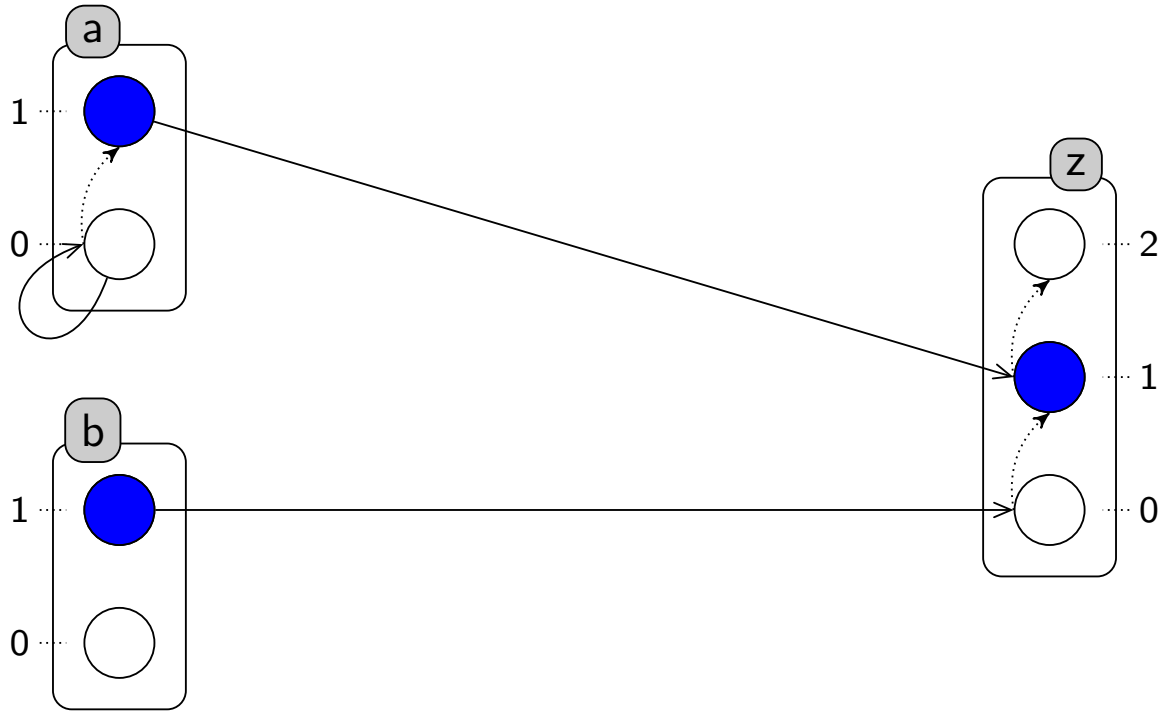
Sorts: components $a,\ b,\ z$

Processes: local states / levels of expression $z_0,\ z_1,\ z_2$

States: sets of active processes $\langle a_0, b_1, z_1 \rangle$

Actions: dynamics $b_1 \longrightarrow z_0 \Rsh z_1,\ a_0 \longrightarrow a_0 \Rsh a_1,\ a_1 \longrightarrow z_1 \Rsh z_2$

The Process Hitting modeling



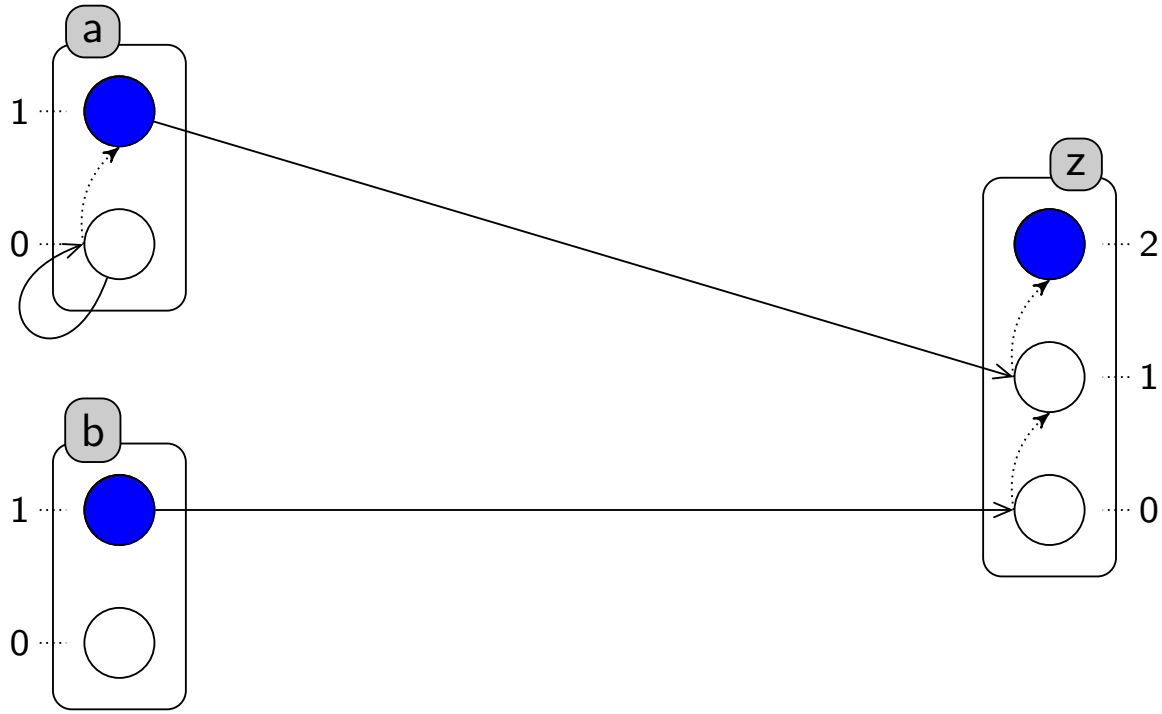
Sorts: components $a,\ b,\ z$

Processes: local states / levels of expression $z_0,\ z_1,\ z_2$

States: sets of active processes $\langle a_1, b_1, z_1 \rangle$

Actions: dynamics $b_1 \longrightarrow z_0 \restriction z_1,\ a_0 \longrightarrow a_0 \restriction a_1,\ a_1 \longrightarrow z_1 \restriction z_2$

The Process Hitting modeling



Sorts: components $a,\ b,\ z$

Processes: local states / levels of expression $z_0,\ z_1,\ z_2$

States: sets of active processes $\langle a_1, b_1, z_2 \rangle$

Actions: dynamics $b_1 \longrightarrow z_0 \restriction z_1,\ a_0 \longrightarrow a_0 \restriction a_1,\ a_1 \longrightarrow z_1 \restriction z_2$

Answer Set Programming

Answer Set Programming (**ASP**):

- Logic program written in language of AnsProlog*
- Form of rules:

$$L_0 \leftarrow L_k, ..., L_m, \textbf{not } L_{m+1}, ..., \textbf{not } L_n.$$

with each L_i : literal in the sense of classical logic.
Rule's meaning:
if L_k,...,L_m are true and if L_{m+1},..., L_n are false then L_0 is true.

Answer Set Programming

Answer Set Programming (**ASP**):

- Logic program written in language of AnsProlog*
- Form of rules:

$$L_0 \leftarrow L_k, ..., L_m, \textbf{ not } L_{m+1}, ..., \textbf{ not } L_n.$$

with each L_i : literal in the sense of classical logic.
Rule's meaning:
if L_k,...,L_m are true and if L_{m+1},..., L_n are false then L_0 is true.

- Constraint:
  $\leftarrow L_k, ..., L_m, \textbf{ not } L_{m+1}, ..., \textbf{ not } L_n.$
- Fact:
  $L_0.$

Answer Set Programming

Exemple:

$bird(X) \leftarrow lays\_egg(X).$
$mammal(X) \leftarrow engender(X).$
$fly(X) \leftarrow bird(X),$ **not** $mammal(X).$
$lays\_egg(tweety).$

Answer Set Programming

Exemple:

$bird(X) \leftarrow lays\_egg(X).$
$mammal(X) \leftarrow engender(X).$
$fly(X) \leftarrow bird(X),$ **not** $mammal(X).$
$lays\_egg(tweety).$

Solution:

$bird(tweety) \leftarrow True.$
$mammal(tweety) \leftarrow unkown.$

Answer Set Programming

Exemple:

$bird(X) \leftarrow lays\_egg(X).$
$mammal(X) \leftarrow engender(X).$
$fly(X) \leftarrow bird(X),$ **not** $mammal(X).$
$lays\_egg(tweety).$

Solution:

$bird(tweety) \leftarrow True.$
$mammal(tweety) \leftarrow unkown.$
$fly(tweety) \leftarrow bird(tweety),$ **not** $mammal(tweety).$

Answer Set Programming

Exemple:

$bird(X) \leftarrow lays\_egg(X).$
$mammal(X) \leftarrow engender(X).$
$fly(X) \leftarrow bird(X),$ **not** $mammal(X).$
$lays\_egg(tweety).$

Solution:

$bird(tweety) \leftarrow True.$
$mammal(tweety) \leftarrow unkown.$
$fly(tweety) \leftarrow bird(tweety),$ **not** $mammal(tweety).$
$fly(tweety) \leftarrow True,$ **not** $unknown.$

Answer Set Programming

Exemple:

$bird(X) \leftarrow lays\_egg(X).$
$mammal(X) \leftarrow engender(X).$
$fly(X) \leftarrow bird(X),$ **not** $mammal(X).$
$lays\_egg(tweety).$

Solution:

$bird(tweety) \leftarrow True.$
$mammal(tweety) \leftarrow unkown.$
$fly(tweety) \leftarrow bird(tweety),$ **not** $mammal(tweety).$
$fly(tweety) \leftarrow True,$ **not** $unknown.$
$fly(tweety) \leftarrow True, True.$

Answer Set Programming

Exemple:

$bird(X) \leftarrow lays\_egg(X).$
$mammal(X) \leftarrow engender(X).$
$fly(X) \leftarrow bird(X),$ **not** $mammal(X).$
$lays\_egg(tweety).$

Solution:

$bird(tweety) \leftarrow True.$
$mammal(tweety) \leftarrow unkown.$
$fly(tweety) \leftarrow bird(tweety),$ **not** $mammal(tweety).$
$fly(tweety) \leftarrow True,$ **not** $unknown.$
$fly(tweety) \leftarrow True, True.$
$fly(tweety) \leftarrow True.$

Answer Set Programming

Exemple:

$bird(X) \leftarrow lays\_egg(X).$
$mammal(X) \leftarrow engender(X).$
$fly(X) \leftarrow bird(X),$ **not** $mammal(X).$
$lays\_egg(tweety).$

Solution:

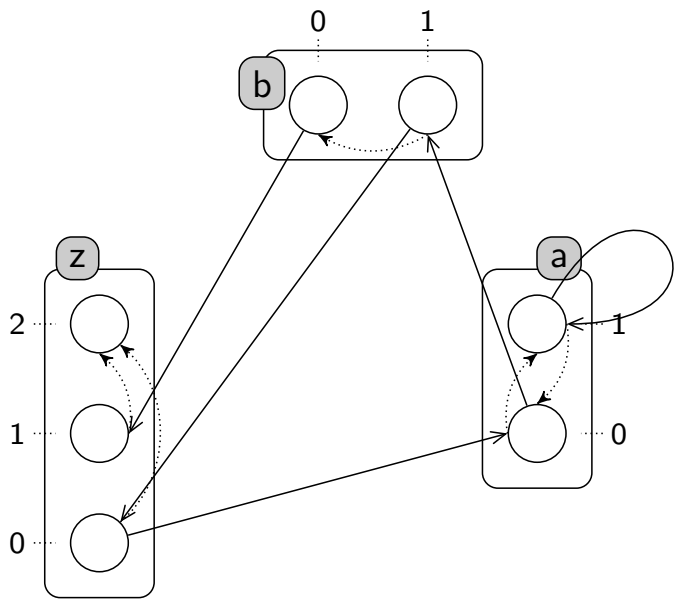
$bird(tweety) \leftarrow True.$
$mammal(tweety) \leftarrow unkown.$
$fly(tweety) \leftarrow bird(tweety),$ **not** $mammal(tweety).$
$fly(tweety) \leftarrow True,$ **not** $unknown.$
$fly(tweety) \leftarrow True, True.$
$fly(tweety) \leftarrow True.$

Answer: `fly(tweety), bird(tweety).`

Fixed Points

Fixed point = state where no action can be fired

→ avoid couples of processes bounded by an action

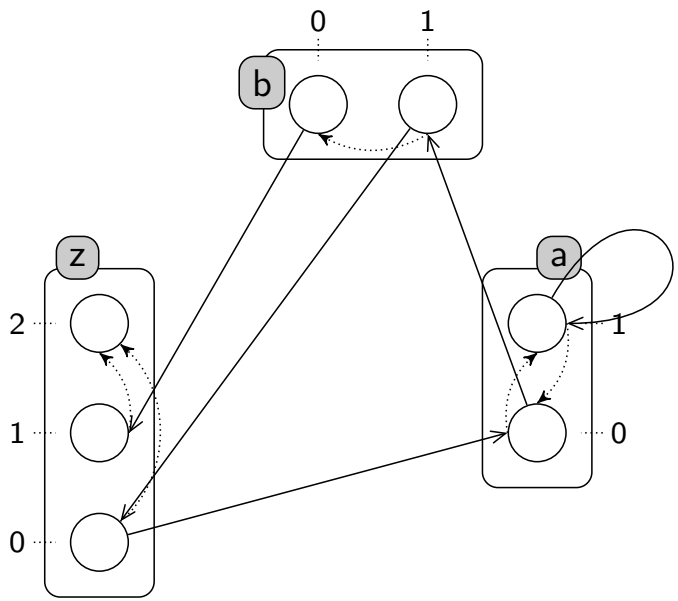


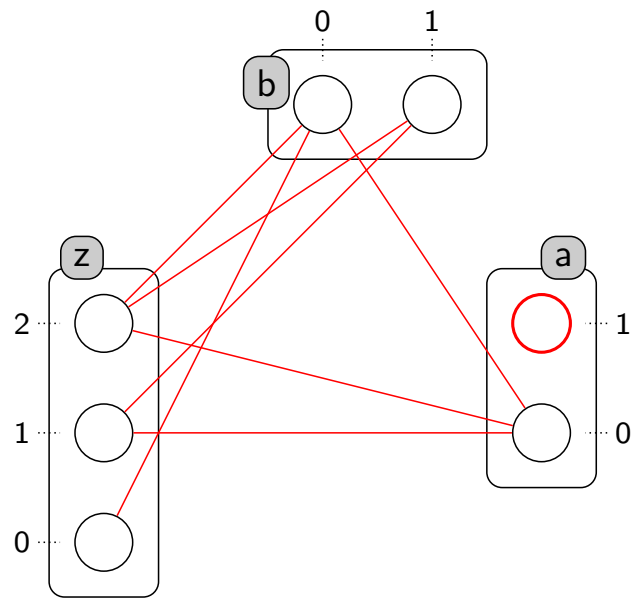
Fixed Points

Fixed point = state where no action can be fired

→ avoid couples of processes bounded by an action

→ Hitless Graph



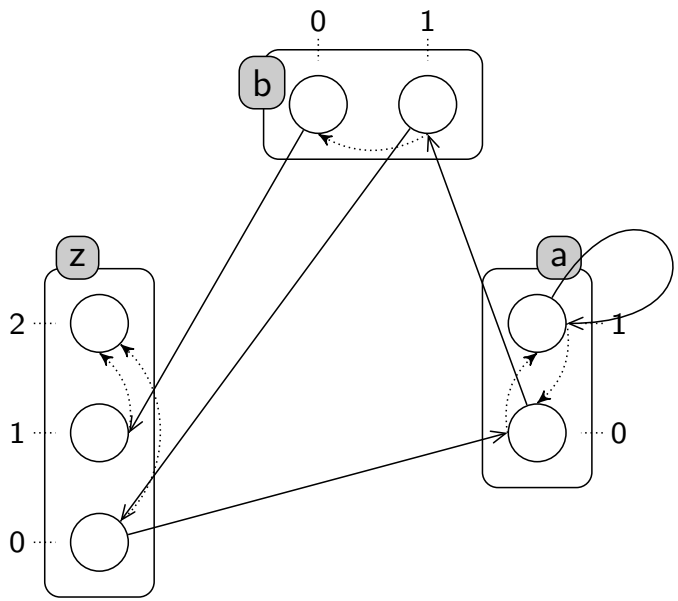


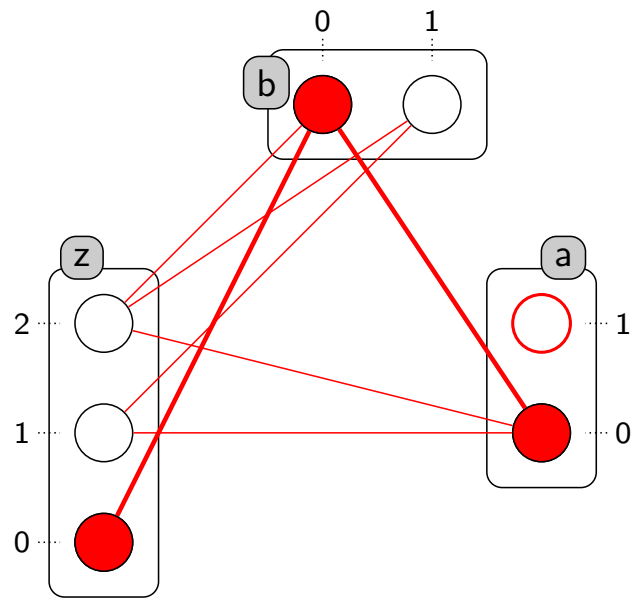
Fixed Points

Fixed point = state where no action can be fired

→ avoid couples of processes bounded by an action

→ Hitless Graph → **n-cliques** = fixed points



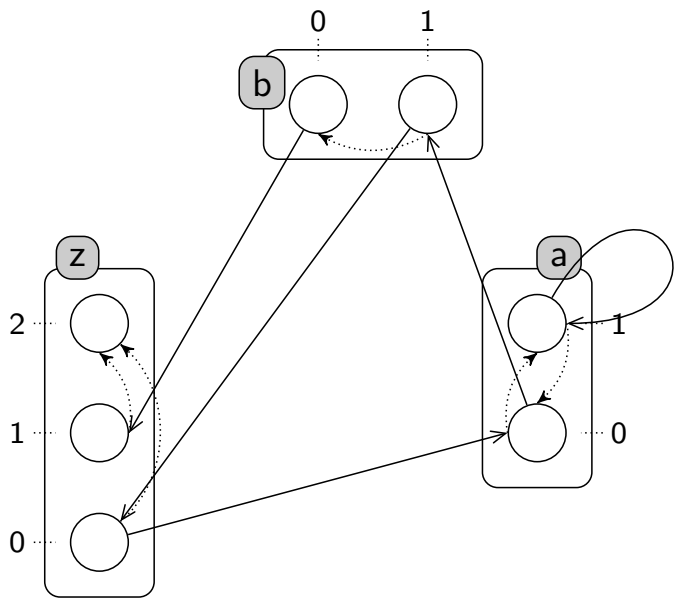


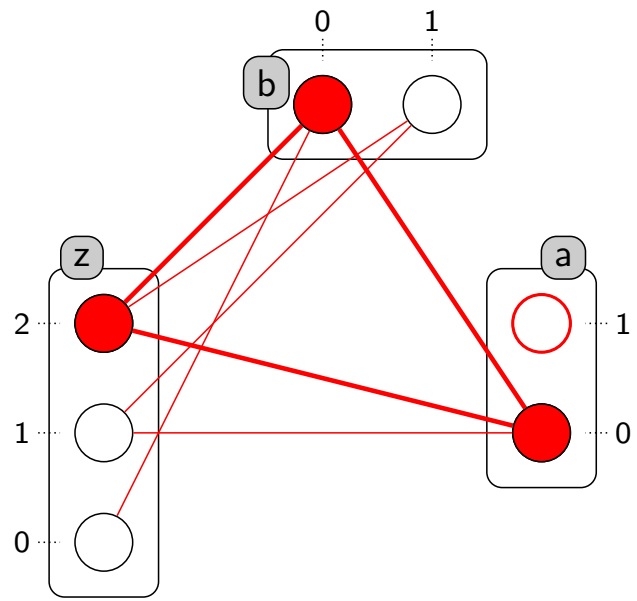
Fixed Points

Fixed point = state where no action can be fired

→ avoid couples of processes bounded by an action

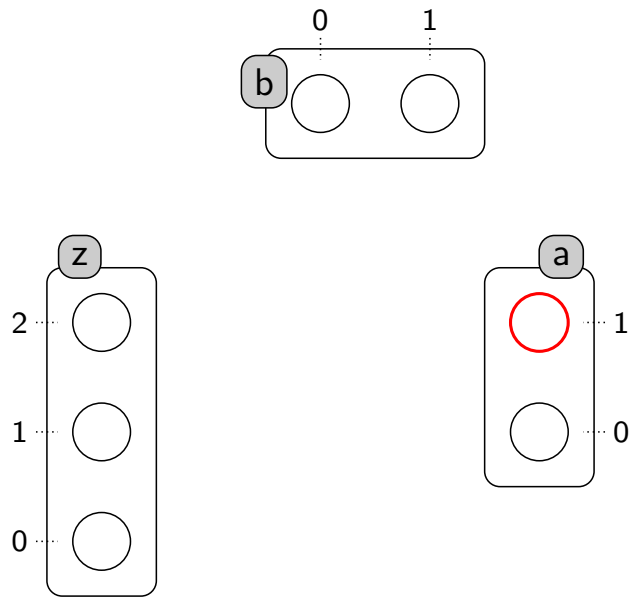
→ Hitless Graph → **n-cliques** = fixed points





Fixed points through ASP

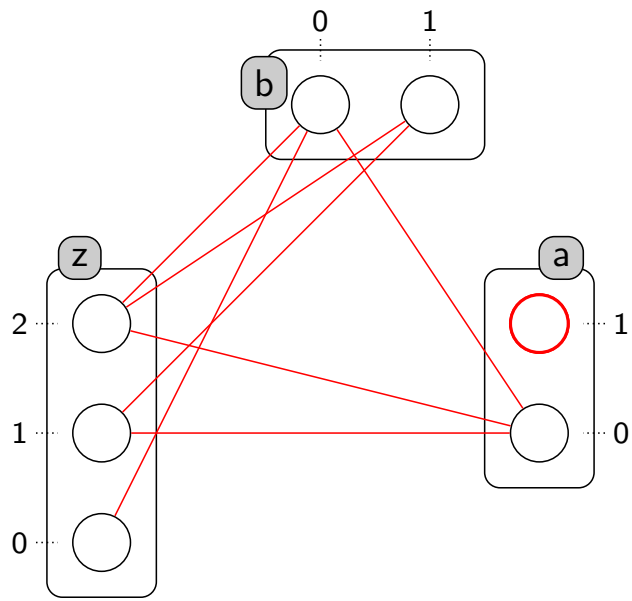
Implementation of algorithm



Fixed points through ASP

Construct of hitless graph

```
noAction(A,I,B,J) :- not hit(A,I,B,J), not hit(B,J,A,I), A!=B,
                     showProcess(A,I), showProcess(B,J).
```



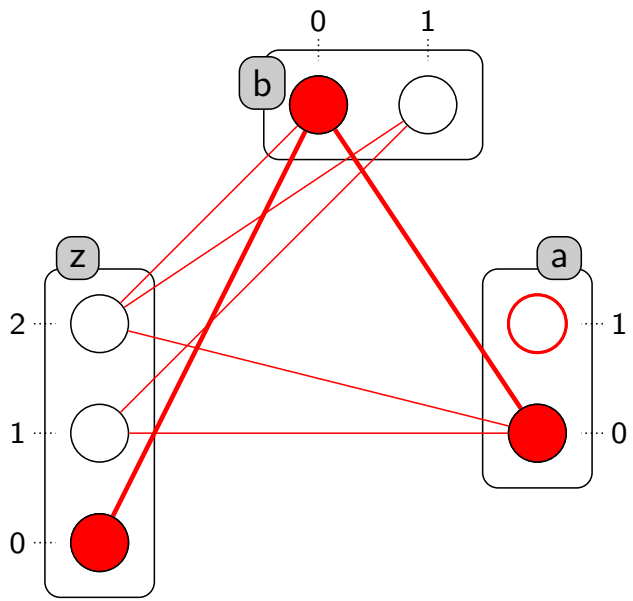
Fixed points through ASP

Construct of hitless graph

```
noAction(A,I,B,J) :- not hit(A,I,B,J), not hit(B,J,A,I), A!=B,
                     showProcess(A,I), showProcess(B,J).
```

Select processes

```
1 {selectProcess(A,I) : showProcess(A,I) } 1 :- sort(A).
```



Fixed points through ASP

Construct of hitless graph

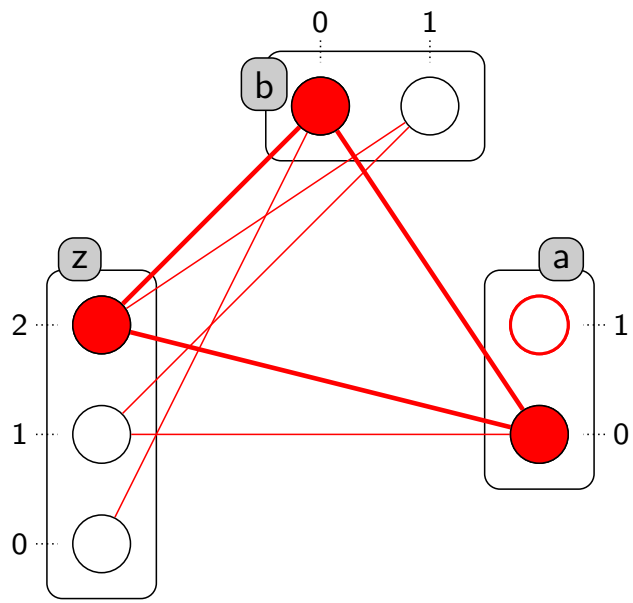
```
noAction(A,I,B,J) :- not hit(A,I,B,J), not hit(B,J,A,I), A!=B,
                     showProcess(A,I), showProcess(B,J).
```

Select processes

```
1 {selectProcess(A,I) : showProcess(A,I) } 1 :- sort(A).
```

Find the fixed points

```
noExistFixPoint :- X<N, getNumberNoHit(X), N={ sort(_) }.
                :- noExistFixPoint.
```



Fixed points through ASP

Construct of hitless graph

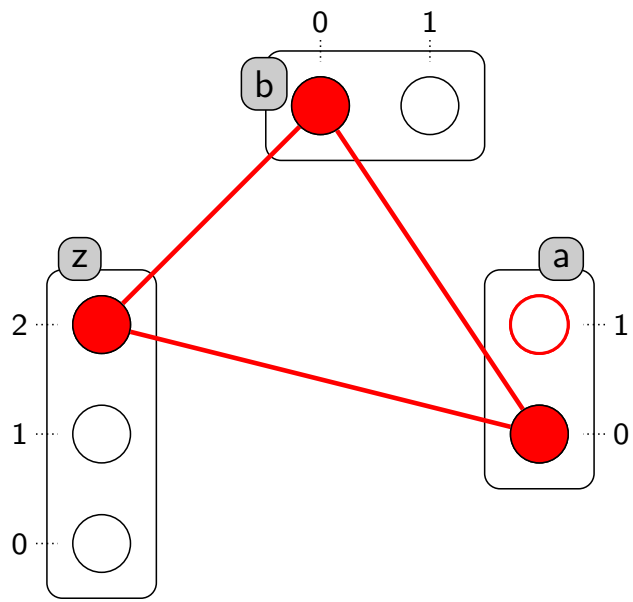
```
noAction(A,I,B,J) :- not hit(A,I,B,J), not hit(B,J,A,I), A!=B,
                     showProcess(A,I), showProcess(B,J).
```

Select processes

```
1 {selectProcess(A,I) : showProcess(A,I) } 1 :- sort(A).
```

Find the fixed points

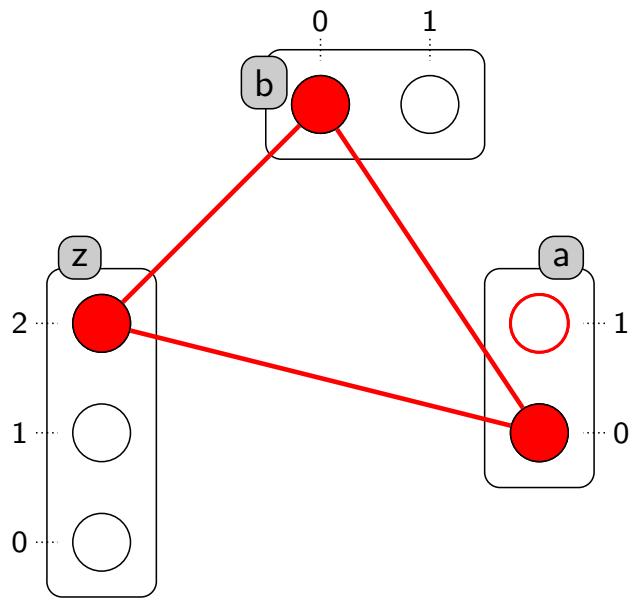
```
noExistFixPoint :- X<N, getNumberNoHit(X), N={ sort(_) }.
                :- noExistFixPoint.
fixProcess(A,I) :- selectProcess(A,I).
```



Fixed points through ASP

ASP program answer:

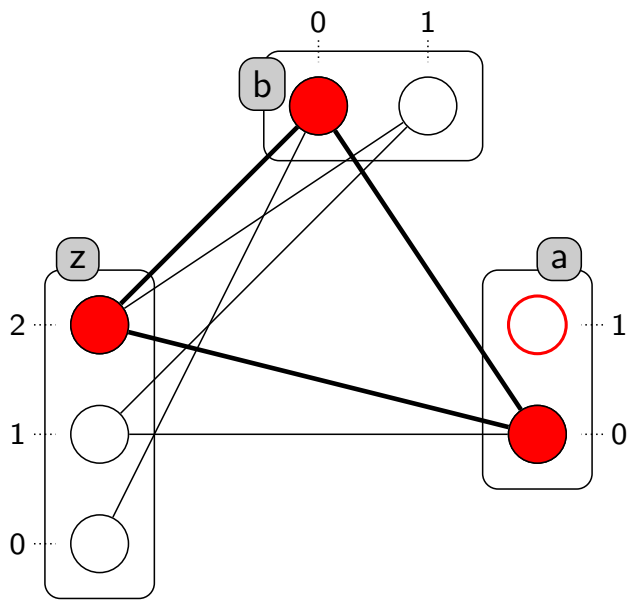
```
Answer 1:  fixProcess(a,0), fixProcess(b,0), fixProcess(z,2).
```



Fixed points through ASP

Optimization:

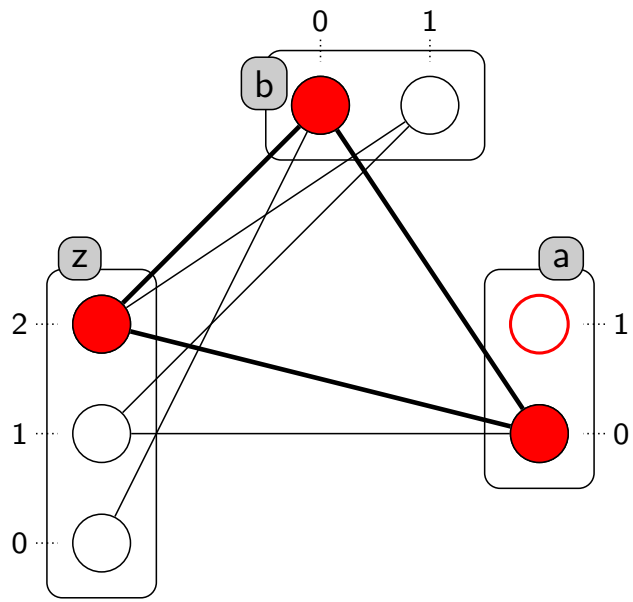
```
1 {selectProcess(A,I) : showProcess(A,I) } 1 :- sort(A).
noHit(A,I,B,J) :- noAction(A,I,B,J), selectProcess(A,I),
                  selectProcess(B,J), A!=B.
noExistFixPoint :- X<N, getNumberNoHit(X), N={ sort(_) }.
                :- noExistFixPoint.
fixProcess(A,I) :- selectProcess(A,I).
```



Fixed points through ASP

Optimization:

```
1 {selectProcess(A,I) : showProcess(A,I) } 1 :- sort(A).
:- 1{hit(A,I,B,J)}, selectProcess(A,I), selectProcess(B,J), A!=B.
fixProcess(A,I) :- selectProcess(A,I).
```



Fixed points through ASP

Comparaison

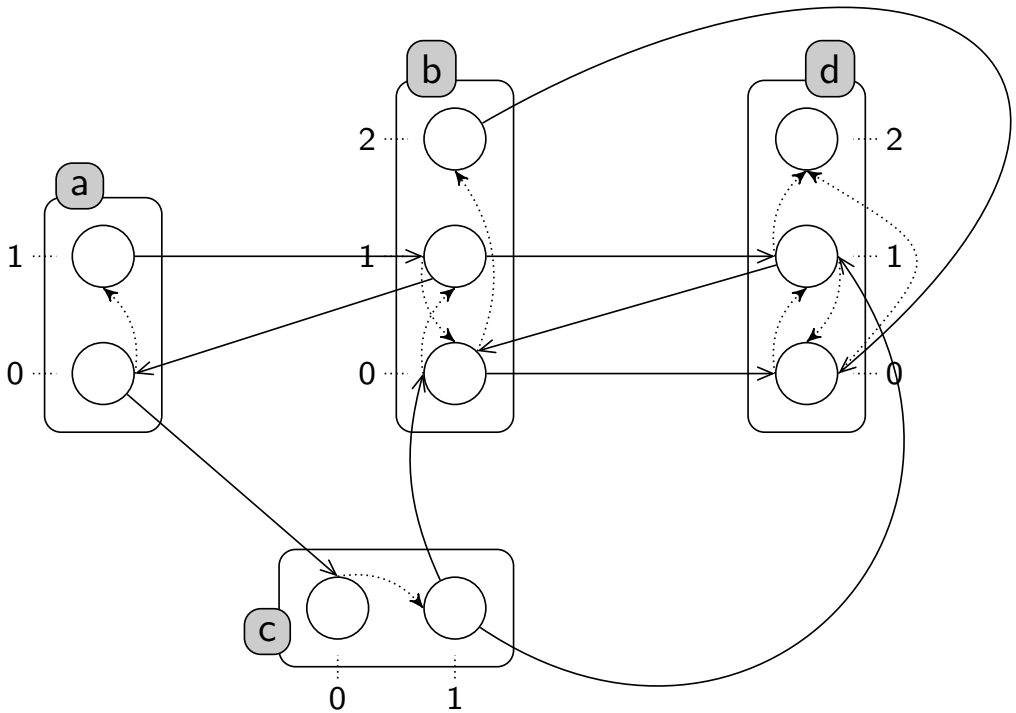
Model	#sorts	#states	#fix-point	PINT	ASP mthd1	mthd2
mvbrn	3	12	1	0.006s	0.000s	0.000s
ERBB_G	42	2^{70}	3	0.017s	0.220s	0.000s
tcrsig40	54	2^{73}	1	0.021s	0.220s	0.020s
tcrsig94	133	2^{194}	0	0.027s	2.540s	0.060s

Figure: Excecution time of ASP metods applied for biological networks with a desktop computer

PINT : a library developed to parse and study PH models

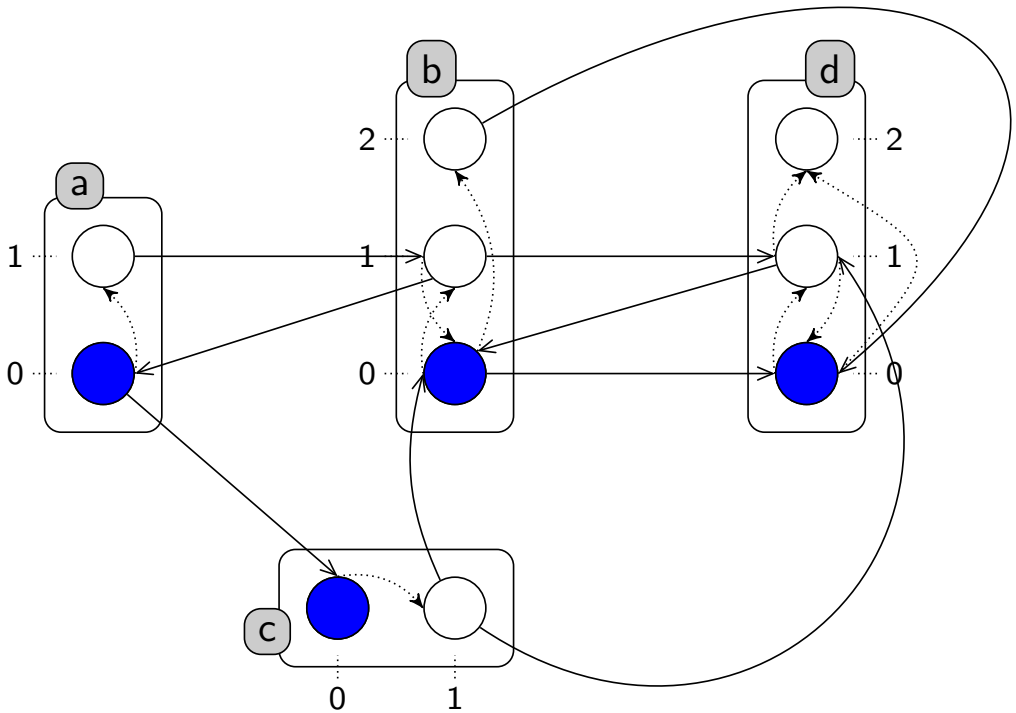
Static analysis: Reachability

Reachability of processes:



Static analysis: Reachability

Reachability of processes:

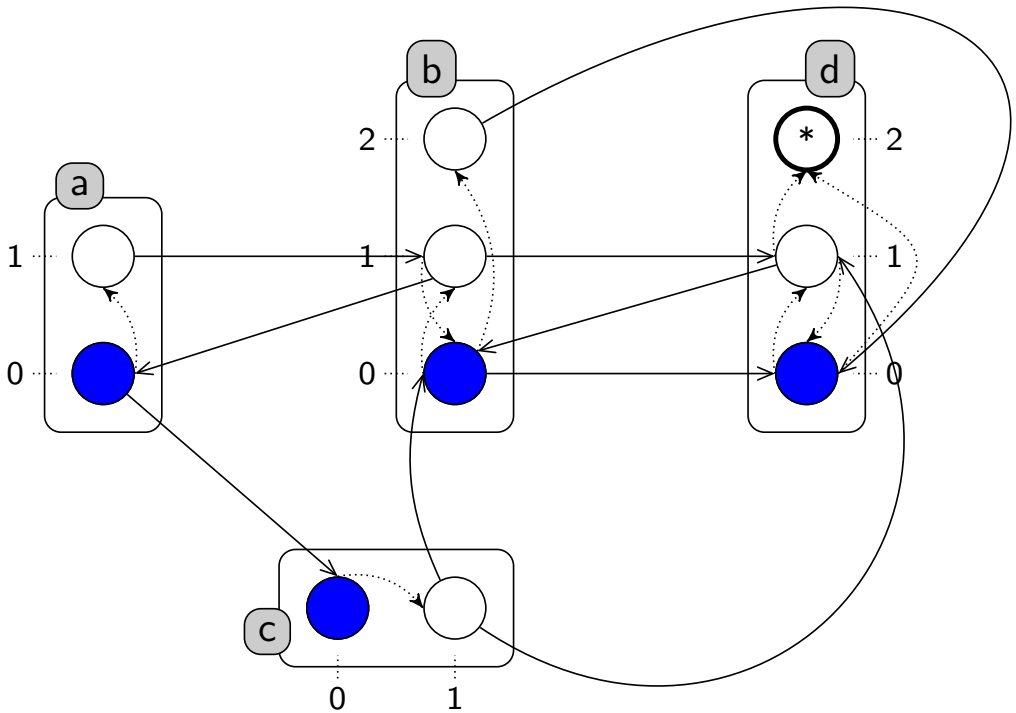


- Initial context

$\langle a_0, b_0, c_0, z_0 \rangle$

Static analysis: Reachability

Reachability of processes:



- Initial context

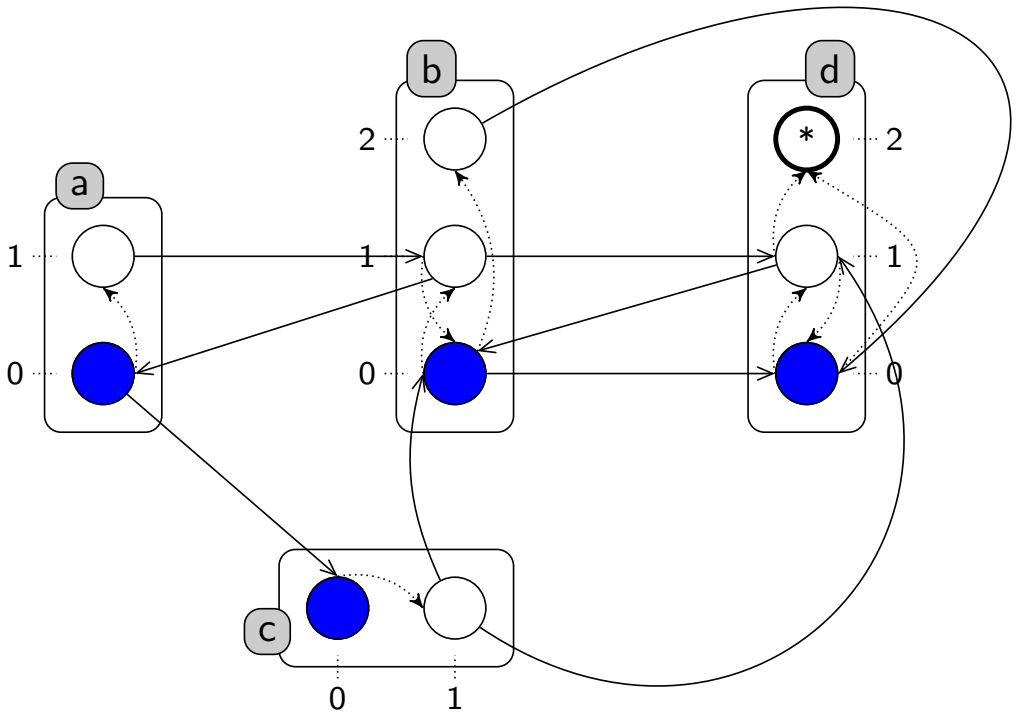
$$\langle a_0, b_0, c_0, z_0 \rangle$$

- Objectives

$$[\ \Rsh\ d_2\ ]$$

Static analysis: Reachability

Reachability of processes:



- Initial context

$$\langle a_0, b_0, c_0, z_0 \rangle$$

- Objectives

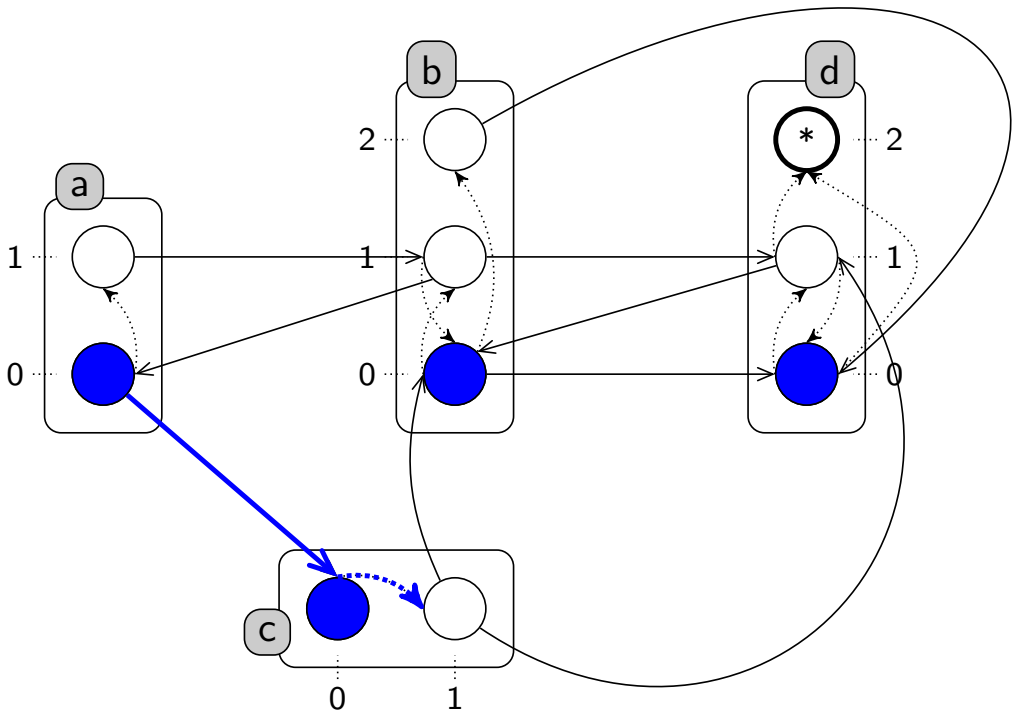
$$[\ \Rsh\ d_2\ ]$$

$\rightarrow$ Concretization of the objective = scenario

$$a_0 \rightarrow c_0 \Rsh c_1 :: \quad b_0 \rightarrow d_0 \Rsh d_1 :: \quad c_1 \rightarrow b_0 \Rsh b_1 :: \quad b_1 \rightarrow d_1 \Rsh d_2$$

Static analysis: Reachability

Reachability of processes:



- Initial context

$$\langle a_0, b_0, c_0, z_0 \rangle$$

- Objectives

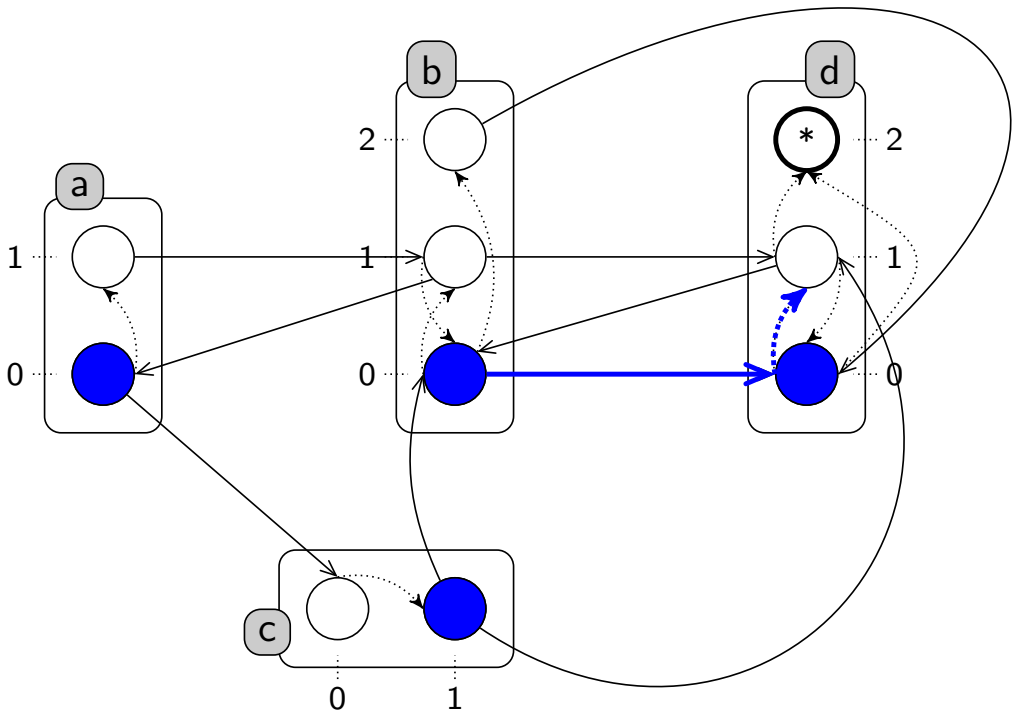
$$[\ \Rsh\ d_2\ ]$$

$\rightarrow$ Concretization of the objective = scenario

$$\underline{a_0 \rightarrow c_0 \Rsh c_1} :: \quad b_0 \rightarrow d_0 \Rsh d_1 :: \quad c_1 \rightarrow b_0 \Rsh b_1 :: \quad b_1 \rightarrow d_1 \Rsh d_2$$

Static analysis: Reachability

Reachability of processes:



- Initial context

$$\langle a_0, b_0, c_0, z_0 \rangle$$

- Objectives

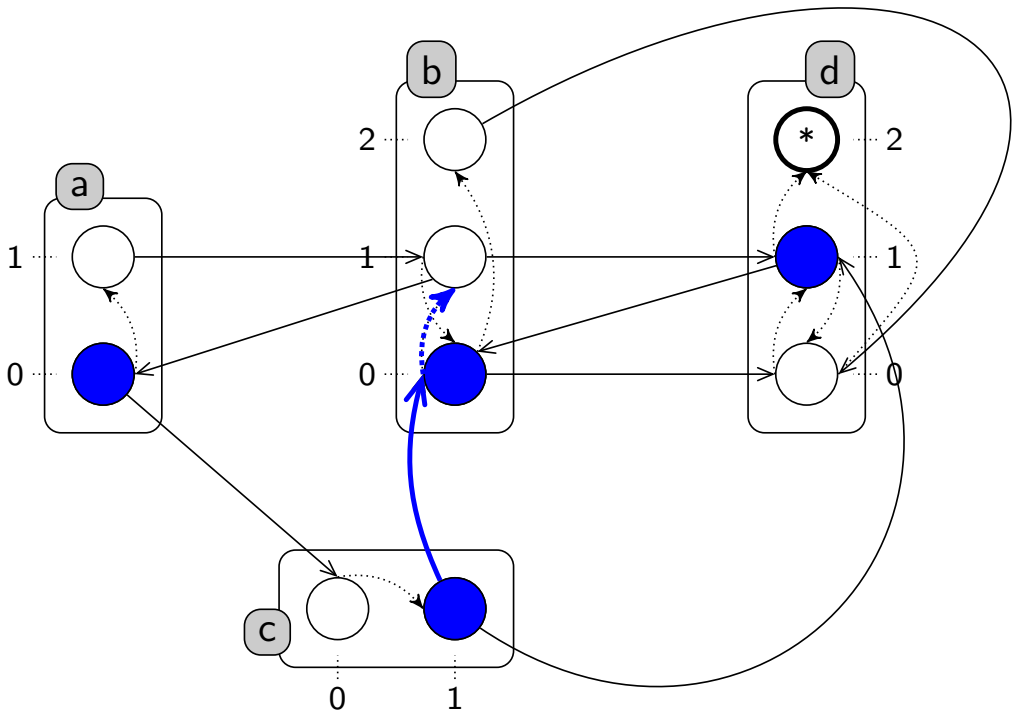
$$[\ \Rsh\ d_2\ ]$$

$\rightarrow$ Concretization of the objective = scenario

$$a_0 \rightarrow c_0 \Rsh c_1 :: \underline{b_0 \rightarrow d_0 \Rsh d_1} :: c_1 \rightarrow b_0 \Rsh b_1 :: b_1 \rightarrow d_1 \Rsh d_2$$

Static analysis: Reachability

Reachability of processes:



- Initial context

$$\langle a_0, b_0, c_0, z_0 \rangle$$

- Objectives

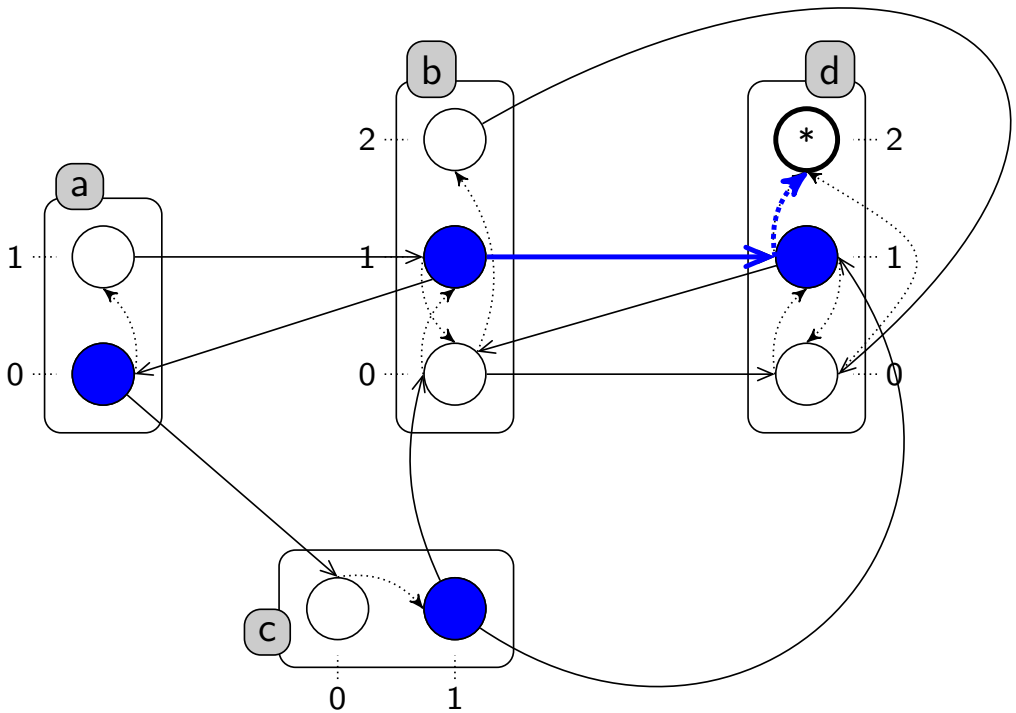
$$[\ \Rsh\ d_2\ ]$$

$\rightarrow$ Concretization of the objective = scenario

$$a_0 \rightarrow c_0 \Rsh c_1 :: \quad b_0 \rightarrow d_0 \Rsh d_1 :: \quad \underline{c_1 \rightarrow b_0 \Rsh b_1} :: \quad b_1 \rightarrow d_1 \Rsh d_2$$

Static analysis: Reachability

Reachability of processes:



- Initial context

$$\langle a_0, b_0, c_0, z_0 \rangle$$

- Objectives

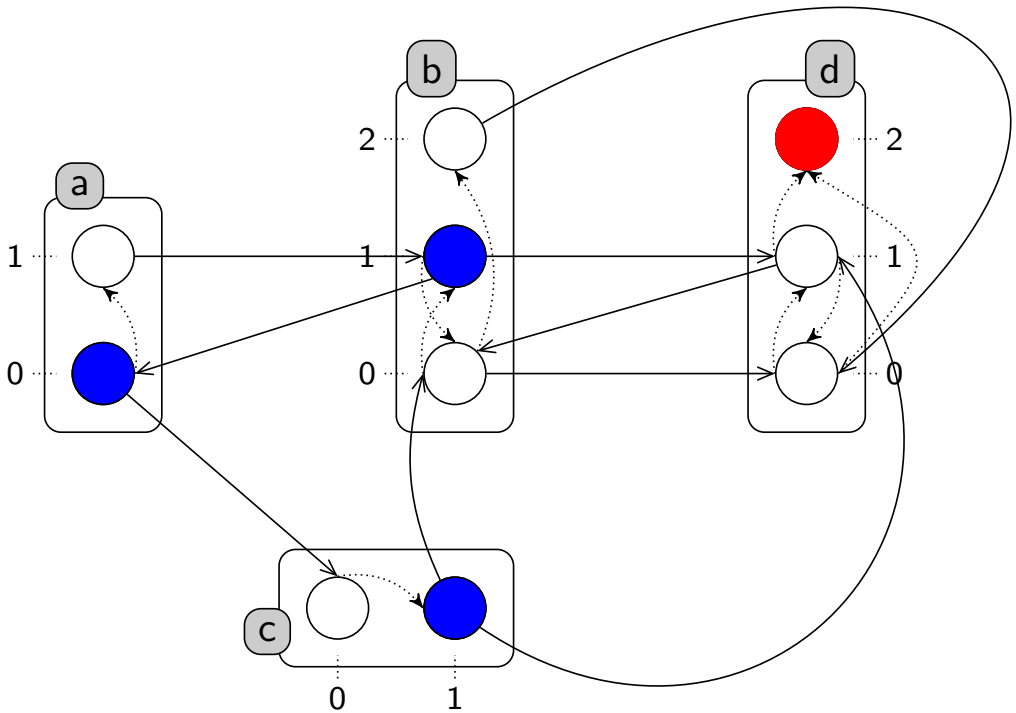
$$[\ \Rsh\ d_2\ ]$$

$\rightarrow$ Concretization of the objective = scenario

$$a_0 \rightarrow c_0 \Rsh c_1 :: \quad b_0 \rightarrow d_0 \Rsh d_1 :: \quad c_1 \rightarrow b_0 \Rsh b_1 :: \quad \underline{b_1 \rightarrow d_1 \Rsh d_2}$$

Static analysis: Reachability

Reachability of processes:



- Initial context

$$\langle a_0, b_0, c_0, z_0 \rangle$$

- Objectives

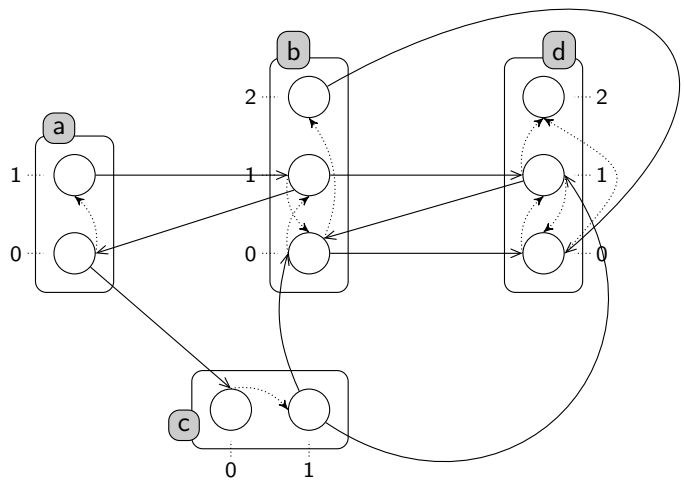
$$[\ \Rsh\ d_2\ ]$$

$\rightarrow$ Concretization of the objective = scenario

$$a_0 \rightarrow c_0 \Rsh c_1 :: \quad b_0 \rightarrow d_0 \Rsh d_1 :: \quad c_1 \rightarrow b_0 \Rsh b_1 :: \quad b_1 \rightarrow d_1 \Rsh d_2$$

Reachability ASP

Network evolution through ASP:

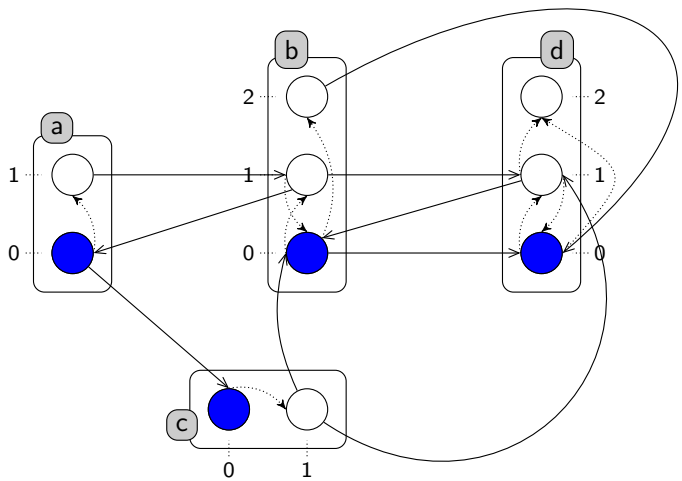


Reachability ASP

Network evolution through ASP:

Initializing :

`init(activeProcess("a",0)).` with `a`: sort, `0`: process index

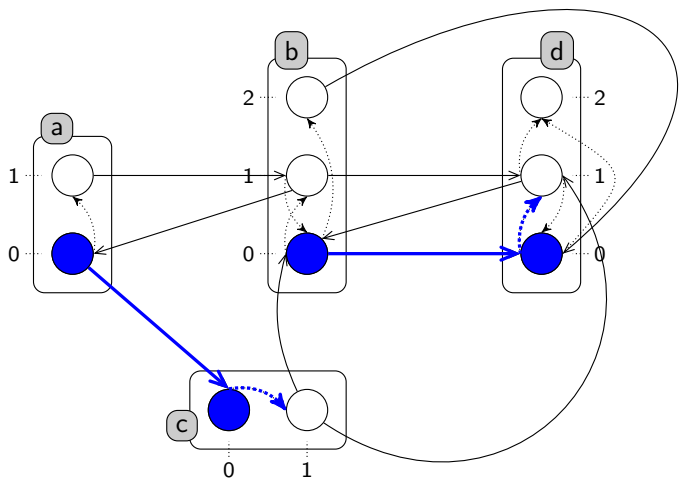


Reachability ASP

Network evolution through ASP:

Playable actions at time T:

```
playableAction(A,I,B,J,K,T) :- action(A,I,B,J,K),
                    instate(actifProcess(B,J),T),
                    instate(actifProcess(A,I),T), time(T).
```

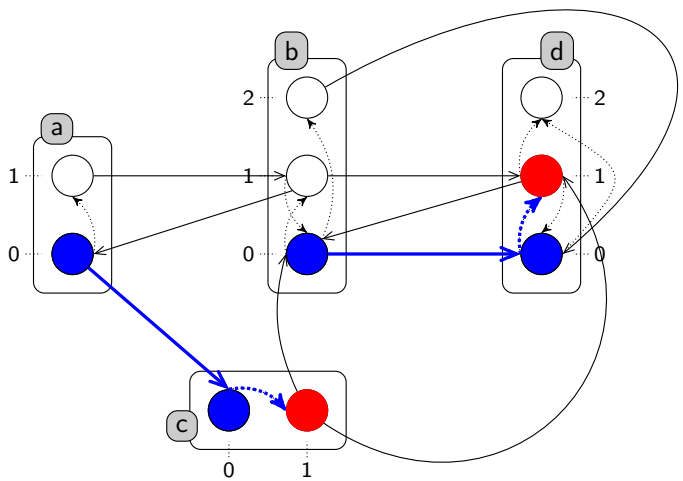


Reachability ASP

Network evolution through ASP:

Change active processes:

```
{activeFromTo(B,J,K,T)} :- playableAction(A,I,B,J,K,T),
                           instate(activeProcess(A,I),T),
                           instate(activeProcess(B,J),T), J!=K, time(T).
                           :- 2{ activeFromTo(B,J,K,T)}, time(T).
```

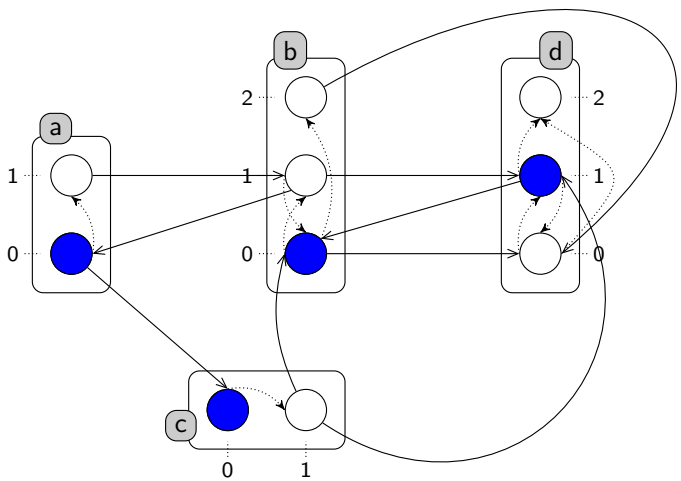


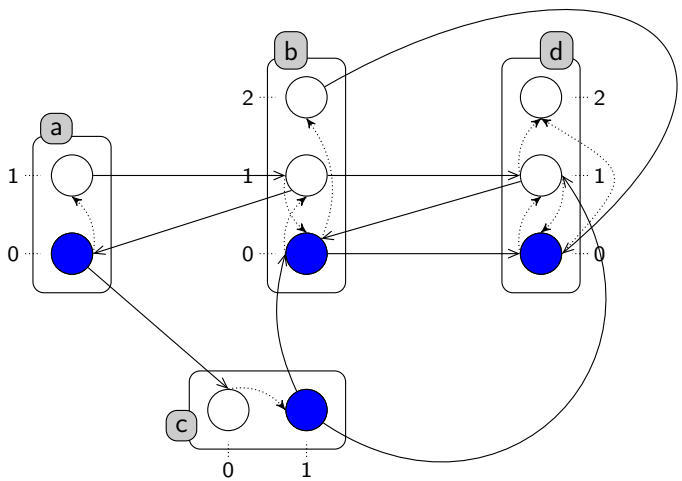
Reachability ASP

Network evolution through ASP:

Active processes at next state (T+1):

```
instate(activeProcess(B,K),T+1) :- activeFromTo(B,J,K,T), time(T).
instate(activeProcess(A,I),T+1) :- instate(activeProcess(A,I),T),
                                   activeFromTo(B,J,K,T), A!=B, time(T).
```

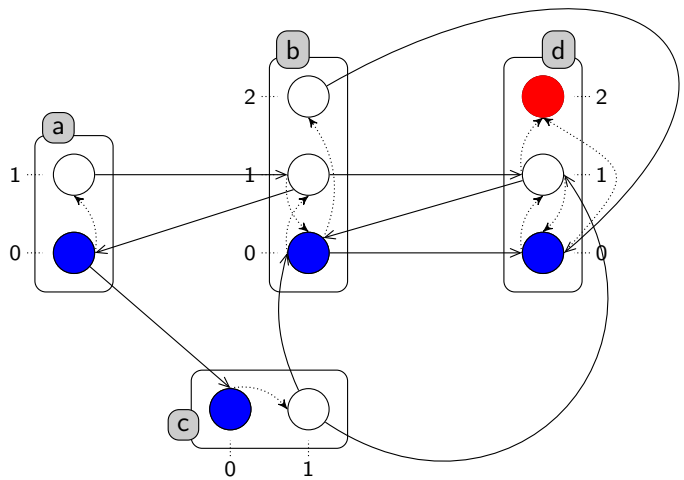




Reachability ASP

Success reachability through ASP:

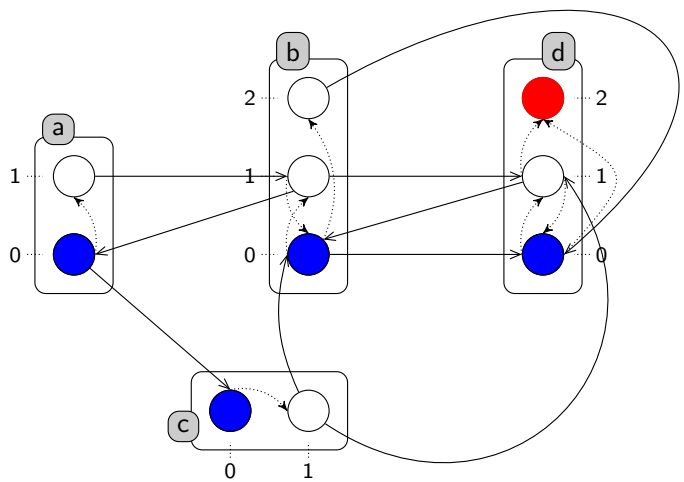
```
goal(activeProcess("d",2)).
```



Reachability ASP

Success reachability through ASP:

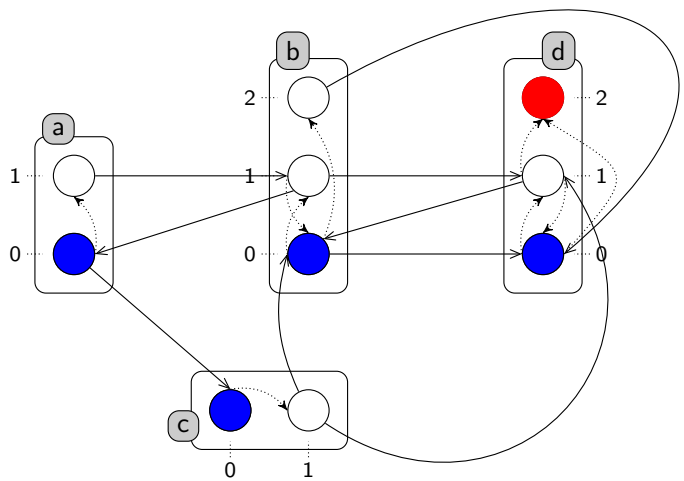
```
goal(activeProcess("d",2)).
satisfaible(F,T) :- goal(F), instate(F,T).
```



Reachability ASP

Success reachability through ASP:

```
goal(activeProcess("d",2)).
satisfaible(F,T) :- goal(F), instate(F,T).
time(0..N).
```

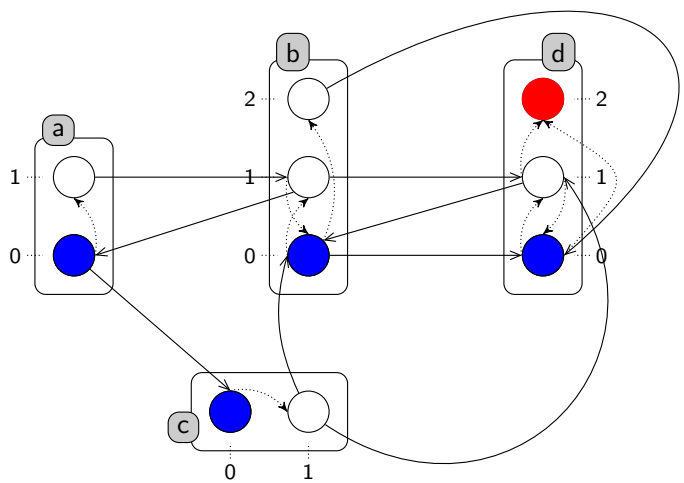


Reachability ASP

Success reachability through ASP:

```
goal(activeProcess("d",2)).
satisfaible(F,T) :- goal(F), instate(F,T).
time(0..N).
```

prédict N –> **Inconvenient**

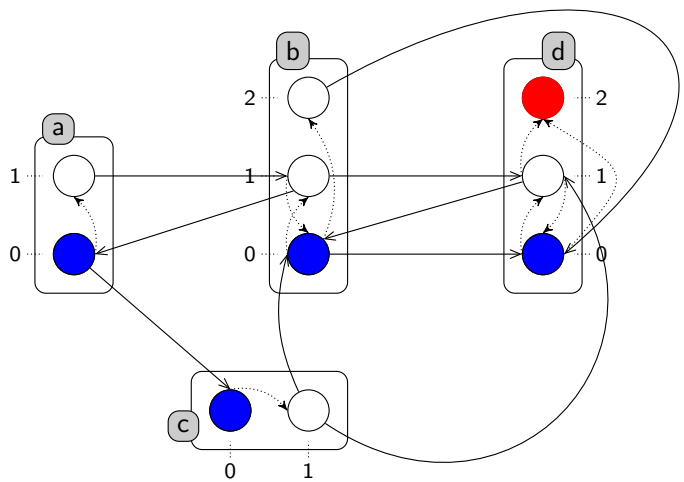


Reachability ASP

Success reachability through ASP:

Iterative:

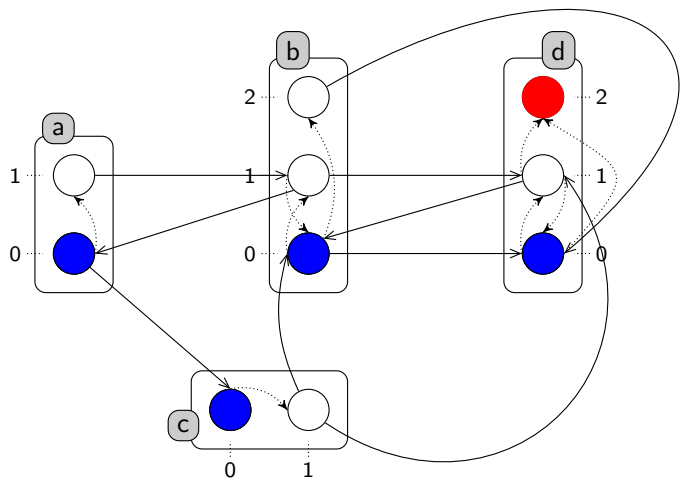
```
goal(activeProcess("d",2)).

#cumulative t
playableAction(A, I, B, J, K,t), activeFromTo(B, J, K,t),
instate(activeProcess(A, I),t + 1)...

#volatile t
notSatisfaible(t) :- goal(F), not instate(F,t).
:- notSatisfaible(t).
```



Reachability ASP

Success reachability through ASP:

Answer 1: `activeFromTo(c,0,1,0), activeFromTo(d,0,1,1), activeFromTo(b,0,1,2), activeFromTo(d,1,2,3).`



Reachability ASP

Comparaison:

ERBB_G1 : **42** sorts, **152** processes, **399** actions, $2^{\mathbf{70}}$*states*
initialisation: level 0 except 5
obective: $goal(activeProcess("pRB", 1))$.

The Table below describes the results:

Model	#steps	PINT	ASP	ASP iterative
ERBB_G1	18	0.022s	10.620s	5.020s

Conclusion

- New developed dynamic analysis to Process hitting models:
  - Fixed point
  - Network evolution
  - Reachability
- Perspectives:
  - Attractors
  - Network evolution without cycles

Bibliography

[1] Loïc Paulevé, Morgan Magnin, Olivier Roux. Modelisation, Simulation et Verification des Grands Reseaux de Regulation Biologique.PhD thesis at École centrale de nantes, 2011.

[2] Baral Chitta. Knowledge representation, reasoning and declarative problem solving. *Cambridge university press*, 2003.

[3] Gebser, Martin and Kaminski, Roland and Kaufmann, Benjamin and Ostrowski, Max and Schaub, Torsten and Thiele, Sven A user's guide to gringo, clasp, clingo, and iclingos, 2008.

Thank you