The NII-Yamanashi-LRI Workshop

Exhaustive analysis of the dynamics of Process Hitting through Answer Set Programming

Emna BEN ABDALLAH & Maxime FOSCHETTE

joint work Olivier ROUX & Morgan MAGNIN

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

Année: 2013/2014

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 did 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 did I do?

   Predicting the **evolutions** of the network.

3) What for?

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

Plan

Plan

Plan

1) Answer set programming (ASP)
   - Definition
   - Example of an ASP program

Plan

1) Answer set programming (ASP)
   - Definition
   - Example of an ASP program
2) Process Hitting (PH)
   - Definition
   - PH through ASP

Plan

1) Answer set programming (ASP)
   - Definition
   - Example of an ASP program
2) Process Hitting (PH)
   - Definition
   - PH through ASP
3) Fixed point
   - Definition
   - ASP implementation
   - Optimization of ASP implementation
   - Comparaison

Plan

1) Answer set programming (ASP)
   - Definition
   - Example of an ASP program
2) Process Hitting (PH)
   - Definition
   - PH through ASP
3) Fixed point
   - Definition
   - ASP implementation
   - Optimization of ASP implementation
   - Comparaison
4) Reachability
   - Definition
   - ASP implementation
   - Iterative ASP implementation
   - Comparaison

Plan

1) Answer set programming (ASP)
   - Definition
   - Example of an ASP program
2) Process Hitting (PH)
   - Definition
   - PH through ASP
3) Fixed point
   - Definition
   - ASP implementation
   - Optimization of ASP implementation
   - Comparaison
4) Reachability
   - Definition
   - ASP implementation
   - Iterative ASP implementation
   - Comparaison
5) Conclusion & prospect

Answer Set Programming

ASP:

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

$$\textit{head} \leftarrow \textit{body}.$$
$$L_0 \leftarrow L_1, ..., L_m, \textbf{not } L_{m+1},..., \textbf{not } L_n.$$

with each L_i : literal in the sense of classical logic.

Rule's meaning:

If L_1,...,L_m are **true** and if L_{m+1},..., L_n are **false**
then L_0 is **true**.

Answer Set Programming

Special types of rules:

- **Constraint** :

$$\leftarrow L_1, ..., L_m, \textbf{not } L_{m+1}, ..., \textbf{not } L_n.$$

- **Fact** :

$$L_0.$$

- **Cardinality** :

$$min\{L_0, ..., L_j\}max \leftarrow L_1, ..., L_m, \textbf{not } L_{m+1}, ..., \textbf{not } L_n.$$

Answer Set Programming

Example:

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

Answer Set Programming

Example:

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

Solution:

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

Answer Set Programming

Example:

$bird(X) \leftarrow lays\_egg(X).$
$mammal(X) \leftarrow engender(X).$
$fly(X) \leftarrow bird(X),$ **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

Example:

$bird(X) \leftarrow lays\_egg(X).$
$mammal(X) \leftarrow engender(X).$
$fly(X) \leftarrow bird(X),$ **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

Example:

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

Solution:

$bird(tweety) \leftarrow True.$
$mammal(tweety) \leftarrow unkown.$
$fly(tweety) \leftarrow bird(tweety), \textbf{not}\ mammal(tweety).$
$fly(tweety) \leftarrow True, \textbf{not}\ unknown.$
$fly(tweety) \leftarrow True, True.$

Answer Set Programming

Example:

$bird(X) \leftarrow lays\_egg(X).$
$mammal(X) \leftarrow engender(X).$
$fly(X) \leftarrow bird(X),$ **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

Example:
$bird(X) \leftarrow lays\_egg(X).$
$mammal(X) \leftarrow engender(X).$
$fly(X) \leftarrow bird(X), \textbf{not } mammal(X).$
$lays\_egg(tweety).$

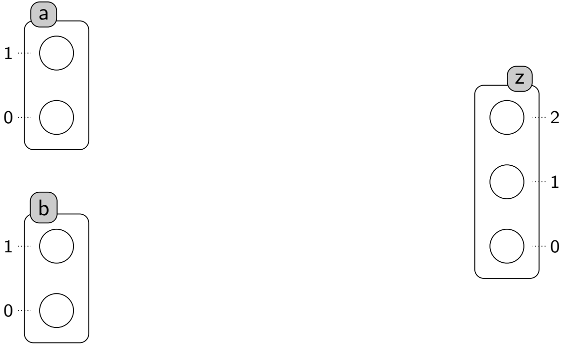
Solution:
$bird(tweety) \leftarrow True.$
$mammal(tweety) \leftarrow unkown.$
$fly(tweety) \leftarrow bird(tweety), \textbf{not } mammal(tweety).$
$fly(tweety) \leftarrow True, \textbf{not } unknown.$
$fly(tweety) \leftarrow True, True.$
$fly(tweety) \leftarrow True.$

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

The Process Hitting modeling

Sorts: components *a*, *b*, *z*

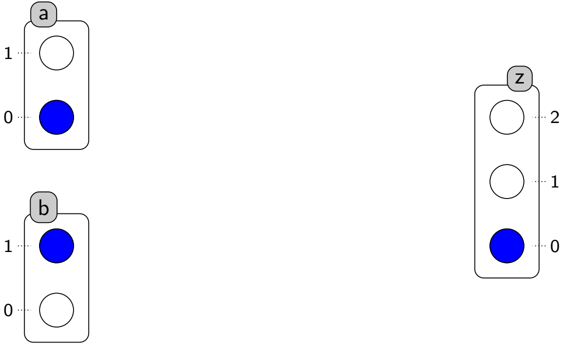
The Process Hitting modeling



Sorts: components a, b, z

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

The Process Hitting modeling

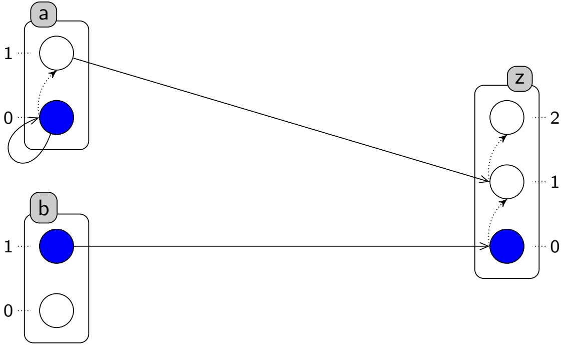


Sorts: components a, b, z

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

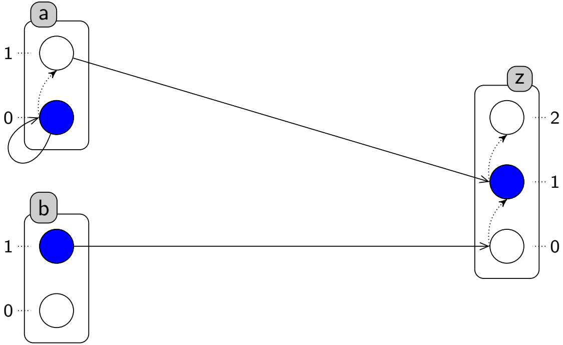
Processes: local states / levels of expression z_0, z_1, z_2

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

Actions: dynamics $b_1 \rightarrow z_0 \upharpoonright z_1$, $a_0 \rightarrow a_0 \upharpoonright a_1$, $a_1 \rightarrow z_1 \upharpoonright z_2$

$\underline{b_1 \rightarrow z_0 \upharpoonright z_1}$, $a_0 \rightarrow a_0 \upharpoonright a_1$, $a_1 \rightarrow z_1 \upharpoonright z_2$

The Process Hitting modeling



Sorts: components a, b, z

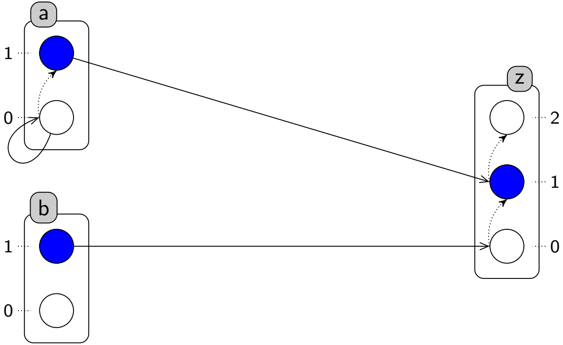
Processes: local states / levels of expression z_0, z_1, z_2

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

Actions: dynamics $b_1 \rightarrow z_0 \upharpoonright z_1,\ a_0 \rightarrow a_0 \upharpoonright a_1,\ a_1 \rightarrow z_1 \upharpoonright z_2$

$b_1 \rightarrow z_0 \upharpoonright z_1,\ \underline{a_0 \rightarrow a_0 \upharpoonright a_1},\ a_1 \rightarrow z_1 \upharpoonright z_2$

The Process Hitting modeling



Sorts: components a, b, z

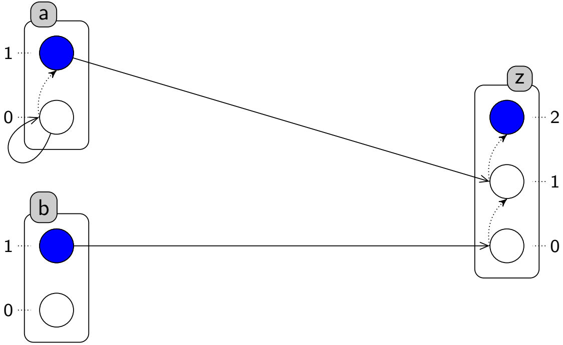
Processes: local states / levels of expression z_0, z_1, z_2

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

Actions: dynamics $b_1 \rightarrow z_0 \upharpoonright z_1$, $a_0 \rightarrow a_0 \upharpoonright a_1$, $a_1 \rightarrow z_1 \upharpoonright z_2$

$b_1 \rightarrow z_0 \upharpoonright z_1$, $a_0 \rightarrow a_0 \upharpoonright a_1$, $\underline{a_1 \rightarrow z_1 \upharpoonright z_2}$

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 $b_1 \rightarrow z_0 \Rsh z_1,\ a_0 \rightarrow a_0 \Rsh a_1,\ a_1 \rightarrow z_1 \Rsh z_2$

$b_1 \rightarrow z_0 \Rsh z_1,\ a_0 \rightarrow a_0 \Rsh a_1,\ a_1 \rightarrow z_1 \Rsh z_2$

PH through ASP

Network traduction:

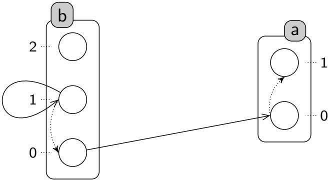
- **Sort**: `sort(A) .`
- **Process**: `process(A,I).`
- **Action** $a_i \rightarrow b_j \Rsh b_k$: `action(A,I,B,J,K).`

PH through ASP

Network traduction:

- **Sort**: `sort(A) .`
- **Process**: `process(A,I).`
- **Action** $a_i \rightarrow b_j \upharpoonright b_k$: `action(A,I,B,J,K).`

Example :

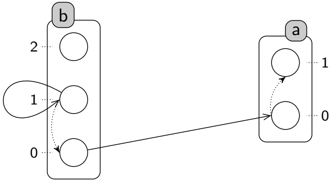


PH through ASP

Network traduction:

- **Sort**: `sort(A) .`
- **Process**: `process(A,I).`
- **Action** $a_i \rightarrow b_j \upharpoonright b_k$: `action(A,I,B,J,K).`

Example :



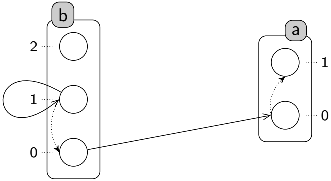
```
sort("a").  sort("b").
```

PH through ASP

Network traduction:

- **Sort**: `sort(A) .`
- **Process**: `process(A,I).`
- **Action** $a_i \rightarrow b_j \upharpoonright b_k$: `action(A,I,B,J,K).`

Example :



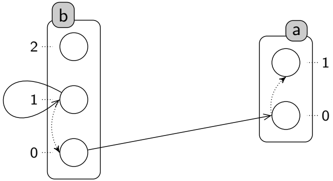
```
sort("a").  sort("b").
process("a", 0..1).
process("b", 0..2).
```

PH through ASP

Network traduction:

- **Sort**: `sort(A) .`
- **Process**: `process(A,I).`
- **Action** $a_i \rightarrow b_j \upharpoonright b_k$: `action(A,I,B,J,K).`

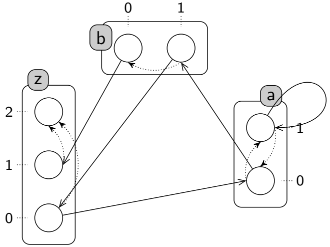
Example :



```
sort("a").  sort("b").
process("a", 0..1).
process("b", 0..2).
action("b",0,"a",0,1).
action("b",1,"b",1,0).
```

Fixed Points

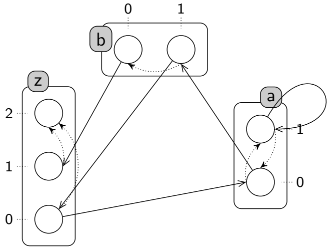
Fixed point = state where no action can be played

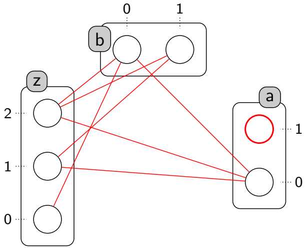


Fixed Points

Fixed point = state where no action can be played

→ Hitless Graph

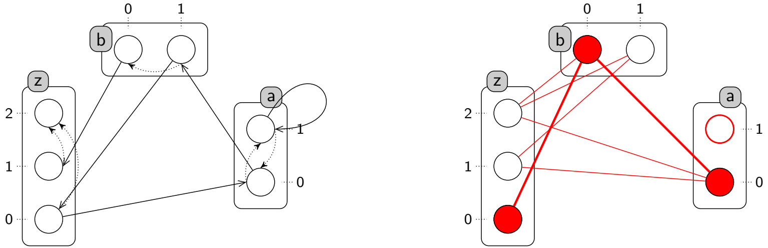




Fixed Points

Fixed point = state where no action can be played

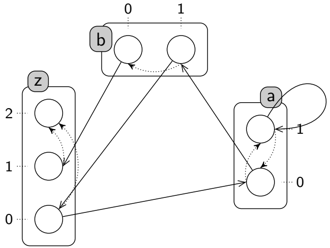
→ Hitless Graph → **n-clics** = fixed points

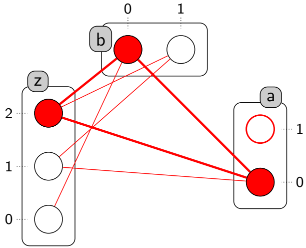


Fixed Points

Fixed point = state where no action can be played

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

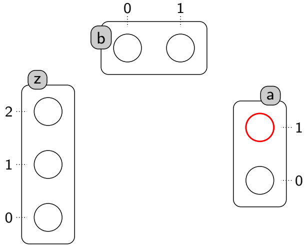




Implementation of the algorithm (N-Cliques)

Definition of hitless graph :

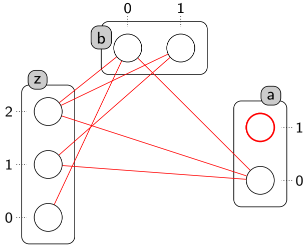
```
noAction(A,I,B,J) :- not hit(A,I,B,J), not hit(B,J,A,I), A!=B,
                     shownProcess(A,I), shownProcess(B,J).
```



Implementation of the algorithm (N-Cliques)

Definition of hitless graph :

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



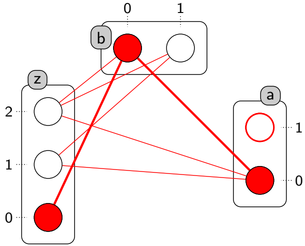
Implementation of the algorithm (N-Cliques)

Definition of hitless graph :

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

Select processes :

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



Implementation of the algorithm (N-Cliques)

Definition of hitless graph :

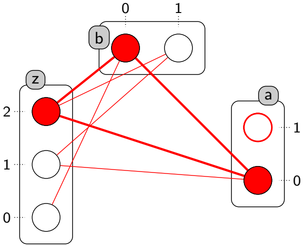
```
noAction(A,I,B,J) :- not hit(A,I,B,J), not hit(B,J,A,I), A!=B,
                     shownProcess(A,I), shownProcess(B,J).
```

Select processes :

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

Find Fixed points :

```
noHit(A,I,B,J) :- noAction(A,I,B,J),
selectProcess(A,I),selectProcess(B,J).
noExistFixPoint :- 0 {noHit(A,I,B,J)} 0, selectProcess(A,I),
                     selectProcess(B,J).
```



Implementation of the algorithm (N-Cliques)

Definition of hitless graph :

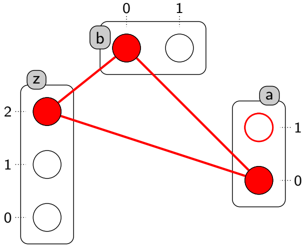
```
noAction(A,I,B,J) :- not hit(A,I,B,J), not hit(B,J,A,I), A!=B,
                     shownProcess(A,I), shownProcess(B,J).
```

Select processes :

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

Find Fixed points :

```
noHit(A,I,B,J) :- noAction(A,I,B,J),
selectProcess(A,I),selectProcess(B,J).
noExistFixPoint :- 0 {noHit(A,I,B,J)} 0, selectProcess(A,I),
                     selectProcess(B,J).
```

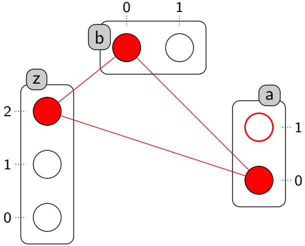


Static analysis

Fixed point through ASP

ASP program result:

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

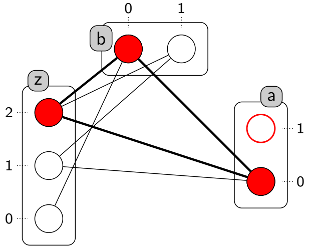


Static analysis

Fixed point 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 :- 0 {noHit(A,I,B,J)} 0, selectProcess(A,I),~~
                     ~~selectProcess(B,J).~~
              ~~:- noExistFixPoint.~~
fixProcess(A,I) :- selectProcess(A,I).
```

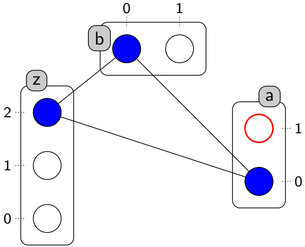


Static analysis

Fixed point through ASP

Optimization:

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



Static analysis

Fixed Point

Comparison

Model	#sorts	#states	#fix-point	mthd1	mthd2	PINT
mvbrn	3	12	1	0.000s	0.000s	0.006s
ERBB	42	2^{70}	3	0.220s	0.000s	0.017s
tcrsig40	54	2^{73}	1	0.220s	0.020s	0.021s
tcrsig94	133	2^{194}	0	2.540s	0.060s	0.027s
egfr104	193	2^{320}	0	8.220s	0.140s	0.074s

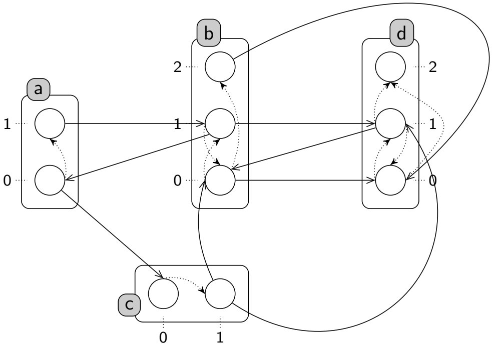
Figure : Excecution time of ASP metods and PINT applied for biological networks with a desktop computer (core i5 and 4GB RAM).

PINT : a library developed to parse and study PH models.

Dynamic analysis

Reachability

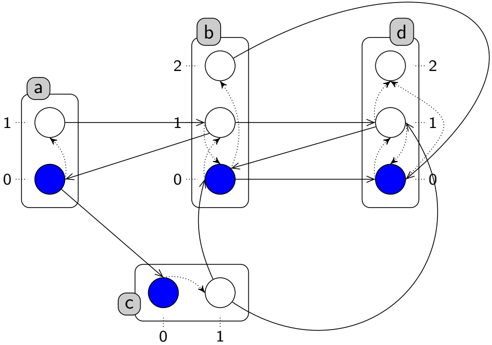
Reachability of processes:



Dynamic analysis

Reachability

Reachability of processes:



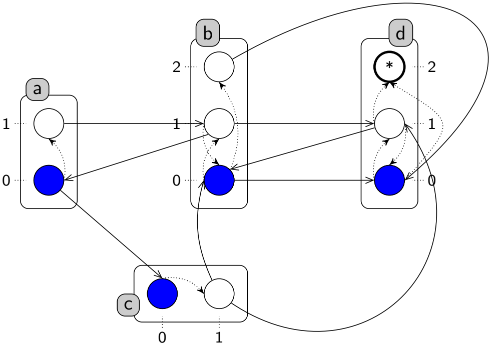
- Initial context

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

Dynamic analysis

Reachability

Reachability of processes:



- Initial context

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

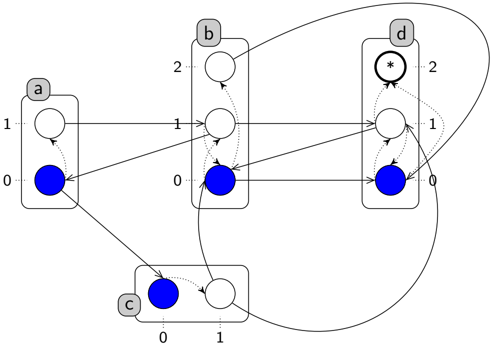
- Objectives

  $[\ \Rsh d_2\ ]$

Dynamic analysis

Reachability

Reachability of processes:



- Initial context

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

- Objectives

  $[\ \Rsh d_2\ ]$

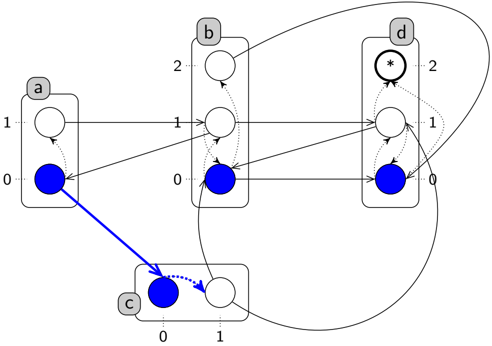
$\rightarrow$ Concretization of the objective = scenario

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

Dynamic analysis

Reachability

Reachability of processes:



- Initial context

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

- Objectives

  $[\Rsh d_2]$

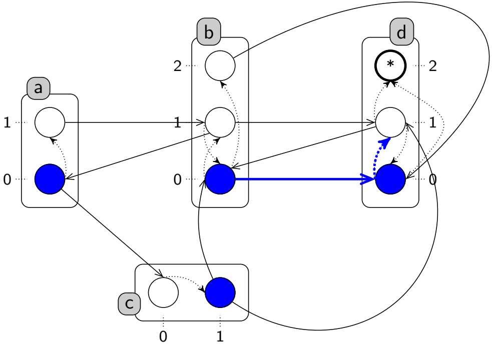
$\rightarrow$ Concretization of the objective = scenario

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

Dynamic analysis

Reachability

Reachability of processes:



- Initial context

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

- Objectives

$[\ \Rsh d_2\ ]$

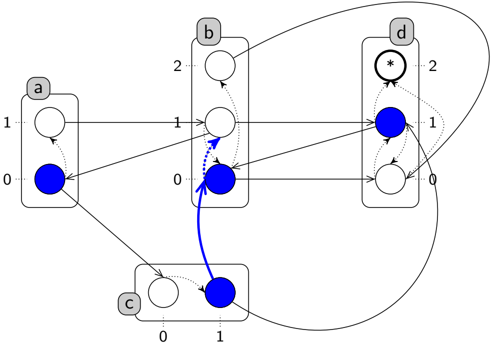
$\rightarrow$ Concretization of the objective = scenario

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

Dynamic analysis

Reachability

Reachability of processes:



- Initial context

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

- Objectives

$[\ \upharpoonright d_2\ ]$

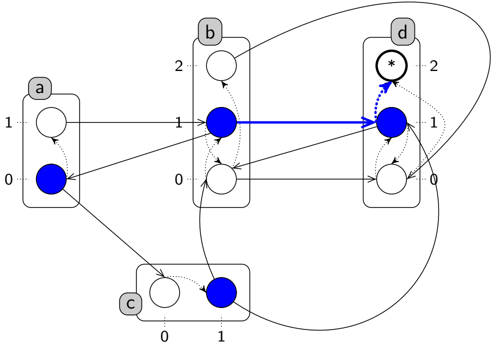
$\rightarrow$ Concretization of the objective = scenario

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

Dynamic analysis

Reachability

Reachability of processes:



- Initial context

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

- Objectives

  $[\ \upharpoonleft d_2\ ]$

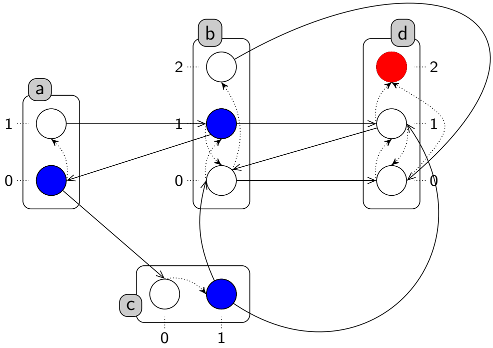
$\rightarrow$ Concretization of the objective = scenario

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

Dynamic analysis

Reachability

Reachability of processes:



- Initial context

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

- Objectives

$[\ \Rsh d_2\ ]$

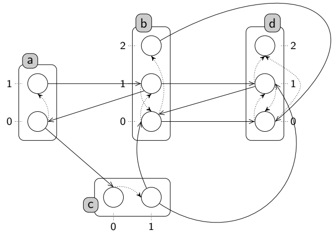
→ Concretization of the objective = scenario

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

Dynamic analysis

Evolution through ASP

Network evolution through ASP



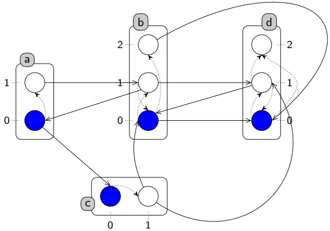
Dynamic analysis

Evolution through ASP

Network evolution through ASP

Initializing :

`init(activeProcess("a",0)).` avec `a:` sorte, `0:` indice du processus



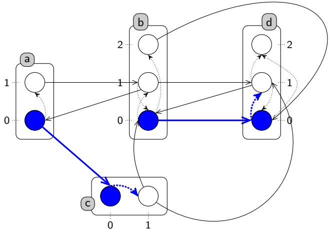
Dynamic analysis

Evolution through ASP

Network evolution through ASP

Playable actions at step T :

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



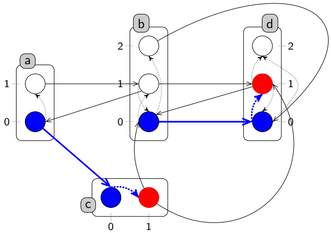
Dynamic analysis

Evolution through ASP

Network evolution through ASP

Change active processes :

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



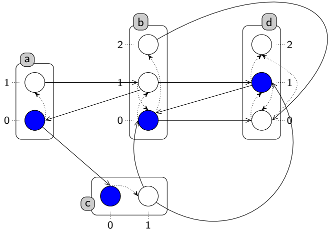
Dynamic analysis

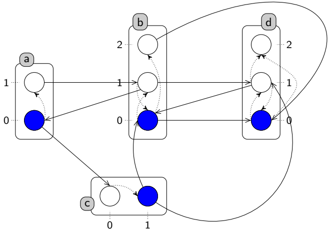
Evolution through ASP

Network evolution through ASP

Active processes at next step (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).
```





Dynamic analysis

Evolution through ASP

Network evolution through ASP

```
time(0..N).
```

Results ($N = 3$) :

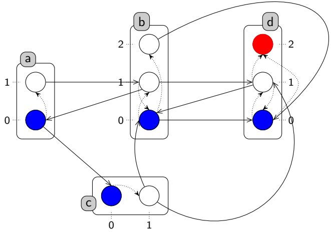
```
Answer 1:   activeFromTo("d",0,1,0) activeFromTo("c",0,1,1)
actifFromTo("b",0,1,2).
Answer 2:   activeFromTo("d",0,1,0) activeFromTo("b",0,2,1)
Answer 3:   activeFromTo("c",0,1,0) activeFromTo("d",0,1,1)
activeFromTo("d",1,0,2) activeFromTo("b",0,1,3)
...
Answer 29:   activeFromTo("c",0,1,0) activeFromTo("b",0,1,1)
activeFromTo("a",0,1,2)
```

Dynamic analysis

Reachability through ASP

Success reachability through ASP:

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

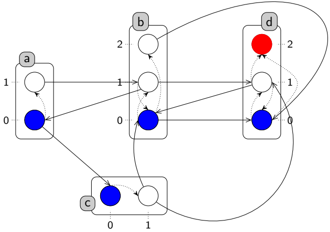


Dynamic analysis

Reachability through ASP

Success reachability through ASP:

```
goal(activeProcess("d",2)).
satisfaible(F,T) :- goal(F), instate(F,T).
:- not satisfaibleTot.
```

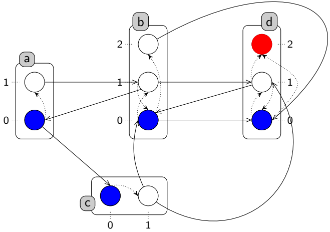


Dynamic analysis

Reachability through ASP

Success reachability through ASP:

```
goal(activeProcess("d",2)).
satisfaible(F,T) :- goal(F), instate(F,T).
:- not satisfaibleTot.
time(0..N).
```



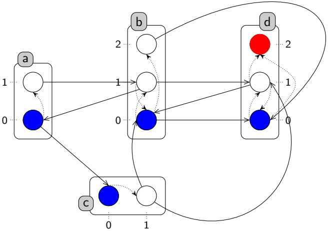
Dynamic analysis

Reachability through ASP

Success reachability through ASP:

```
goal(activeProcess("d",2)).
satisfaible(F,T) :- goal(F), instate(F,T).
:- not satisfaibleTot.
time(0..N).
```

predict N −> Inconvenient



Dynamic analysis

Reachability through ASP

Results for ($N = 2$) :

```
UNSATISFIABLE
```

Results for ($N = 3$) :

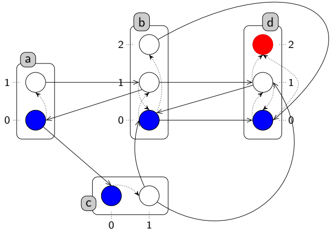
```
Answer 1:   activeFromTo(c,0,1,0), activeFromTo(d,0,1,1),
activeFromTo(b,0,1,2), activeFromTo(d,1,2,3).
Answer 2:   activeFromTo("d",0,1,0) activeFromTo("c",0,1,1)
activeFromTo("b",0,1,2) activeFromTo("d",1,2,3)
```

Dynamic analysis

Reachability through ASP

Success reachability through ASP iterative:

```
goal(activeProcess("d",2)).
#base
instate(F,0) :- init(F).
#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).
```



Dynamic analysis

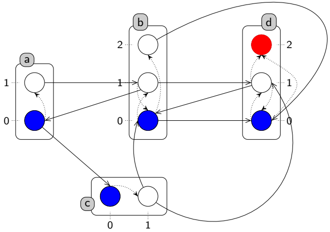
Reachability through ASP

Success reachability through ASP iterative:

Results:

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

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



Dynamic analysis

Reachability through ASP

Comparison:

Initializing biological models components and the objectives.

Model	#sorts	#states	#steps	ASP	ASPi	PINT
Exemple	4	36	4	0.000s	0.000s	0.000s
ERBB	42	2^{70}	18	10.620s	5.020s	0.022s
tcrsig40	54	2^{73}	26	156.500s	127.250s	0.012s

Figure : Excecution time of ASP metods (CLINGO et ICLINGO) and PINT applied for biological networks with a desktop computer (core i5 and 4GB RAM)

Dynamic analysis

Reachability through ASP

Comparison:

Method of Rocca et al.:

- ASP
- CTL properties with model cheking (AF, EF, AG...)
- Transitions graph

Dynamic analysis

Reachability through ASP

Comparison:

Method of Rocca et al.:

- ASP
- CTL properties with model cheking (AF, EF, AG...)
- Transitions graph

Comparaison of the property EF

$$\texttt{prop = EF } (I_0, goal)$$

Dynamic analysis

Reachability through ASP

Comparaison:

Example: Tail resorption of tadpole :
12 sorts, 42 process, 139 actions and 524.288 states.

$$\texttt{prop = EF(}I_0, goal\texttt{)}$$

Dynamic analysis

Reachability through ASP

Comparaison:

Example: Tail resorption of tadpole :
12 sorts, 42 process, 139 actions and 524.288 states.

$$\texttt{prop = EF(}I_0, goal\texttt{)}$$

Network traduction :

- Transition graph: $3min6s$
- Process Hitting : $0.346s$

Dynamic analysis

Reachability through ASP

Comparaison:

Example: Tail resorption of tadpole :
12 sorts, 42 process, 139 actions and 524.288 states.

$$\texttt{prop = EF(}I_0, goal\texttt{)}$$

Network traduction :

- Transition graph: $3min6s$
- Process Hitting : $0.346s$

Property verification :

- Rocca et al. method : $7min17s$
- our itirative method : $1.9s$

Conclusion & Prospects

- New dynamic analysis of Process Hitting models:
  - Fixed point
  - Network evolution
  - Reachability

- Prospects:
  - Adaptation on other models (PN, model of Thomas...)
  - Eliminating cycles
  - Search attractors
  - Reverse reachability ($goal \rightarrow I_0$?)

Bibliography

[1]Christian Anger, Kathrin Konczak, Thomas Linke, and Torsten Schaub. A glimpse of answer set programming. KI, 19(1) :12, 2005.

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

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

[4] Zohra Khalis, Jean-Paul Comet, Adrien Richard, and Gilles Bernot. The smbionet method for discovering models of gene regulatory networks. Genes, Genomes and Genomics, 3(1) :15-22, 2009.

[5] Steffen Klamt, Julio Saez-Rodriguez, Jonathan Lindquist, Luca Simeoni, and Ernst Gilles. A methodology for the structural and functional analysis of signaling and regulatory networks. BMC Bioinformatics, 7(1) :56, 2006.

[6] Vladimir Lifschitz. Answer set programming and plan generation. Artificial Intelligence, 138(1) :39-54, 2002.

Bibliography

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

[8] Loïc Paulevé, Morgan Magnin, and Olivier Roux. Static analysis of biological regulatory net-works dynamics using abstract interpretation. Mathematical Structures in Computer Science, 2012.

[9] Alexandre Rocca, Nicolas Mobilia, Éric Fanchon, Tony Ribeiro, Laurent Trilling, and Katsumi Inoue. Asp for construction and validation of regulatory biological networks. In Luis Fariñas del Cerro and Katsumi Inoue, editors, *Logical Modeling of Biological Systems*, pages 167-206. Wiley-ISTE, 2014.

[10] Regina Samaga, Julio Saez-Rodriguez, Leonidas G Alexopoulos, Peter K. Sorger, and Stef-fen Klamt. The logic of egfr/erbb signaling : Theoretical properties and analysis of high-throughput data. PLoS Computational Biology, 5(8) :e1000438, 2009.

Thanks for your attention