Inference of the limit-language of elementary cellular automata via process graphs and regular expressions

Pedro P.B. de Oliveira
Eurico L.P. Ruivo
Wander L. Costa
Fábio T. Miki
Victor V. Trafaniuc

P.P.B. de Oliveira, Eurico L.P. Ruivo, Wander L. Costa, Fábio T. Miki and Victor V. Trafaniuc.
"Advances in the study of elementary cellular automata regular language complexity"
Complexity 21(6): 267-279, 2016.

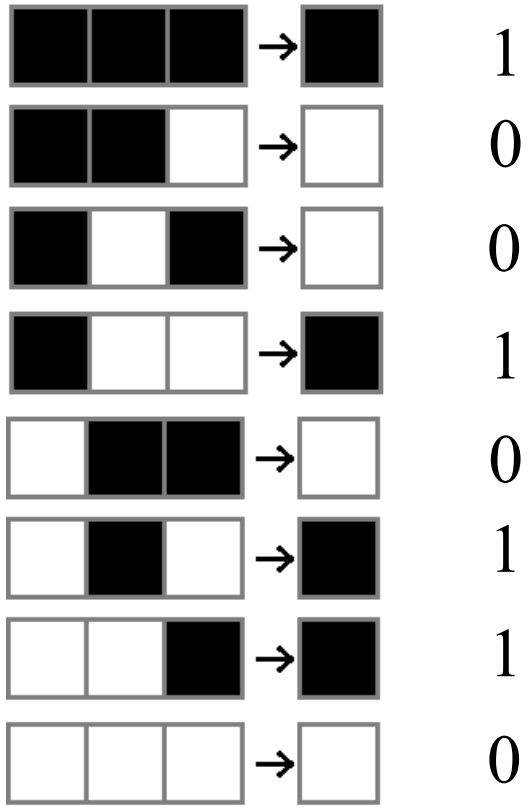
Universidade Presbiteriana Mackenzie

2016

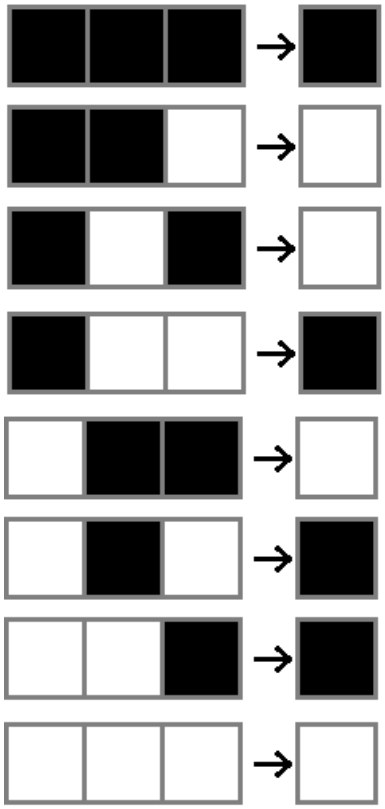
AGENDA

1. **Process graphs of ECAs**
2. **Growth of the ECAs' process graphs**
3. **Covers and difference sets**
4. **Inference of limit graphs**
5. **Results and next steps**

Elementary Cellular Automata (ECAs)

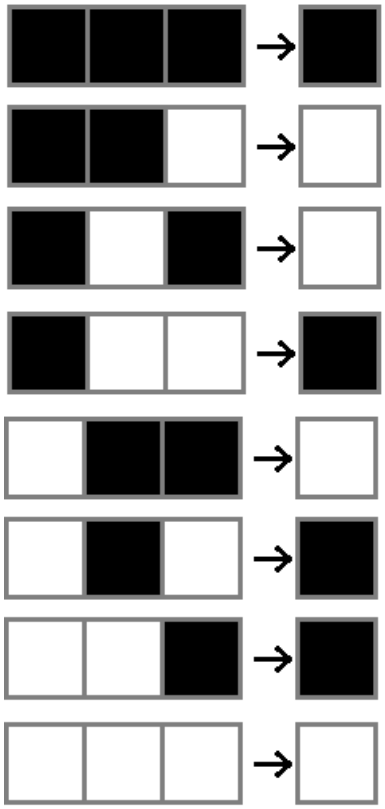


Elementary Cellular Automata (ECAs)



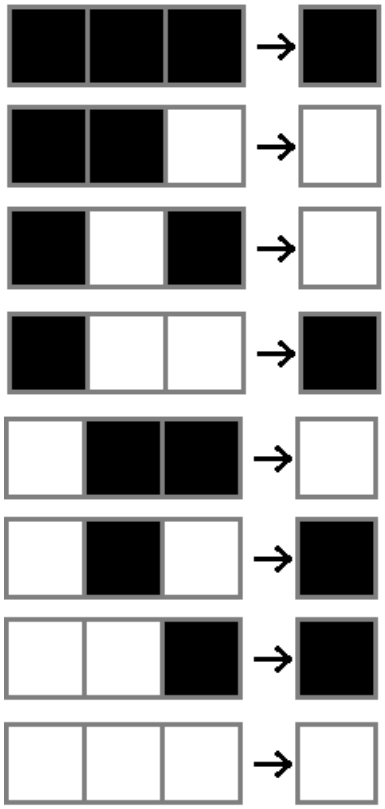
$$1 \times 2^7$$
$$0 \times 2^6$$
$$0 \times 2^5$$
$$1 \times 2^4$$
$$0 \times 2^3$$
$$1 \times 2^2$$
$$1 \times 2^1$$
$$0 \times 2^0$$

Elementary Cellular Automata (ECAs)



$$
\begin{aligned}
1 &\times 2^7 = 128 \\
0 &\times 2^6 = 0 \\
0 &\times 2^5 = 0 \\
1 &\times 2^4 = 16 \\
0 &\times 2^3 = 0 \\
1 &\times 2^2 = 4 \\
1 &\times 2^1 = 2 \\
0 &\times 2^0 = 0
\end{aligned}
$$

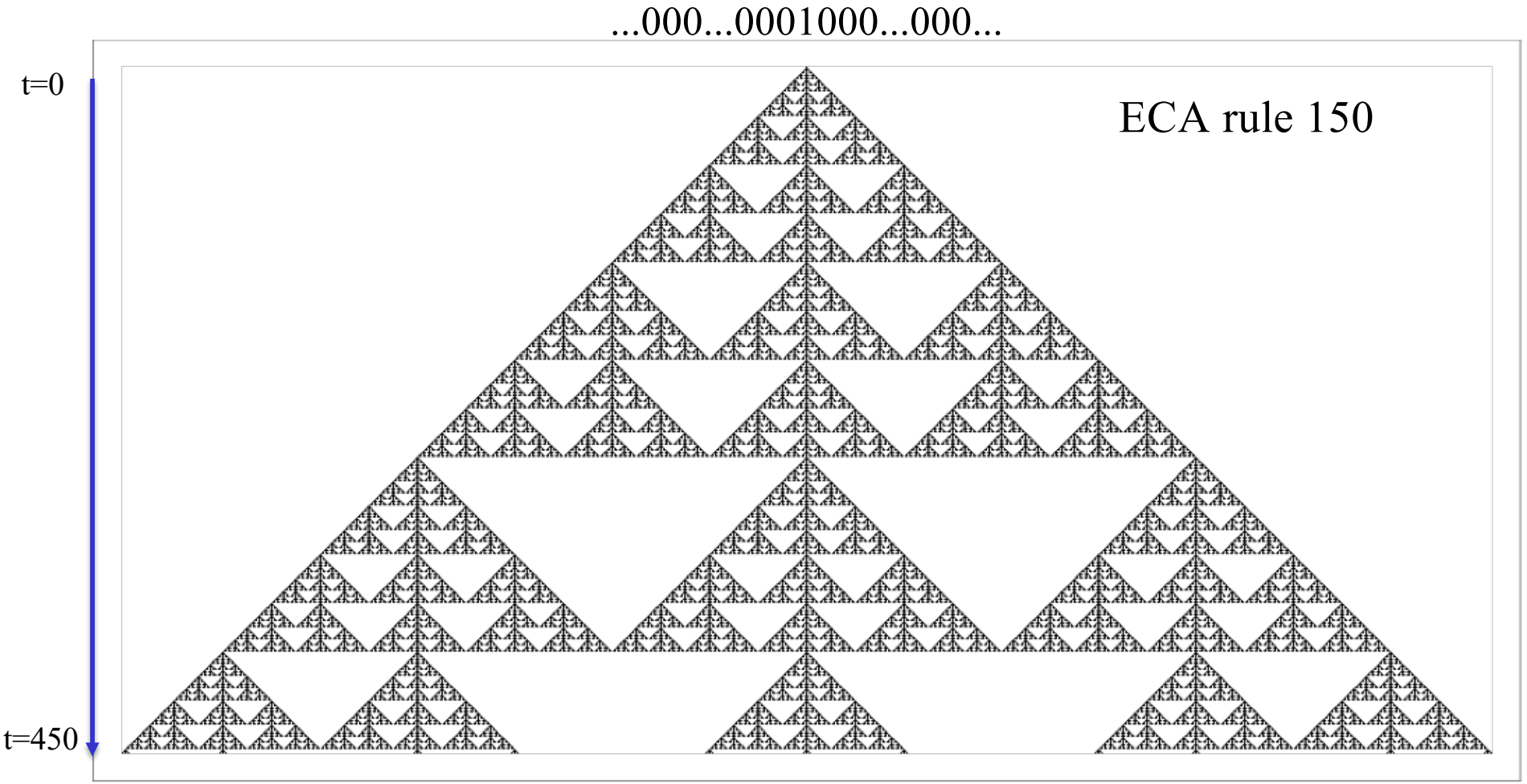
Elementary Cellular Automata (ECAs)



$$
\begin{aligned}
1 &\times 2^7 = 128 \\
0 &\times 2^6 = 0 \\
0 &\times 2^5 = 0 \\
1 &\times 2^4 = 16 \\
0 &\times 2^3 = 0 \\
1 &\times 2^2 = 4 \\
1 &\times 2^1 = 2 \\
0 &\times 2^0 = 0
\end{aligned}
$$

ECA rule 150

Elementary Cellular Automata (ECAs)



1. Process graphs of ECAs

PROCESS GRAPHS OF ECAS

De Bruijn graph

Represents the set of configurations which can be obtained by applying a rule over each possible initial configuration.

f → local ECA rule;
F → global ECA rule induced by f;

Ω^0 → set of all binary configurations;
Ω_f^1 → set of the binary configurations obtained by applying F over each element of Ω^0.

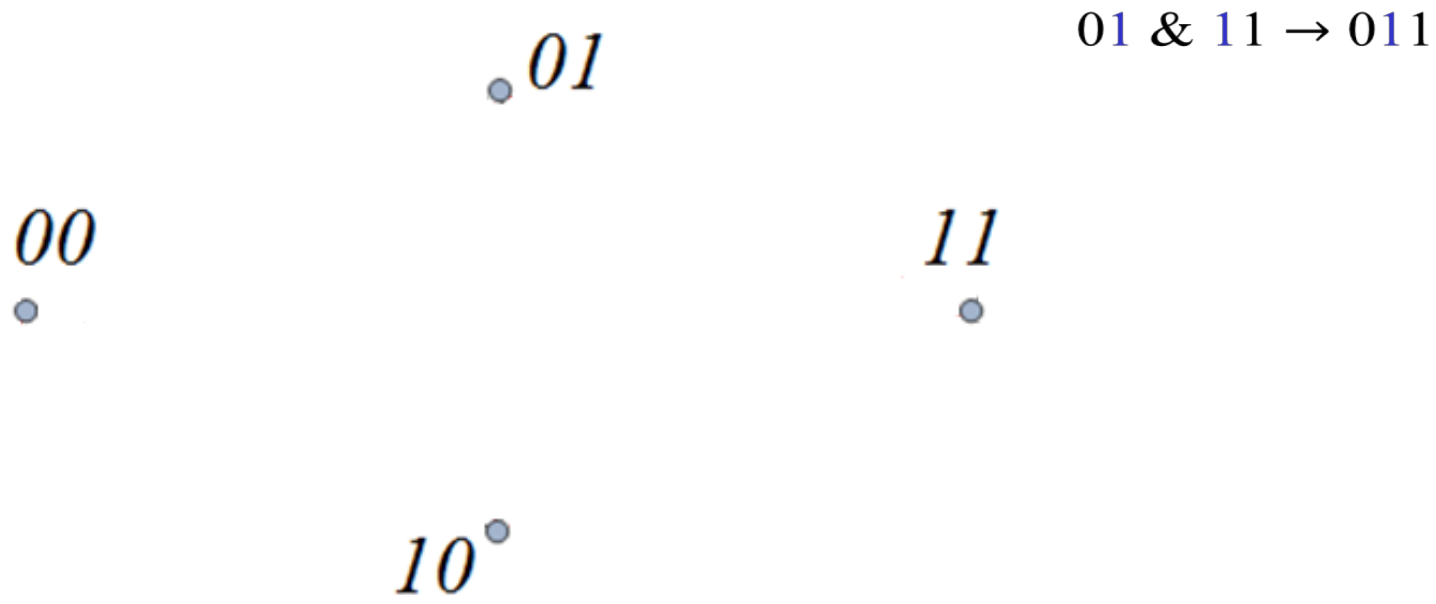
PROCESS GRAPHS OF ECAS

De Bruijn graph

Example (rule 150): $f(x, y, z) = (x + y + z) mod\ 2.$

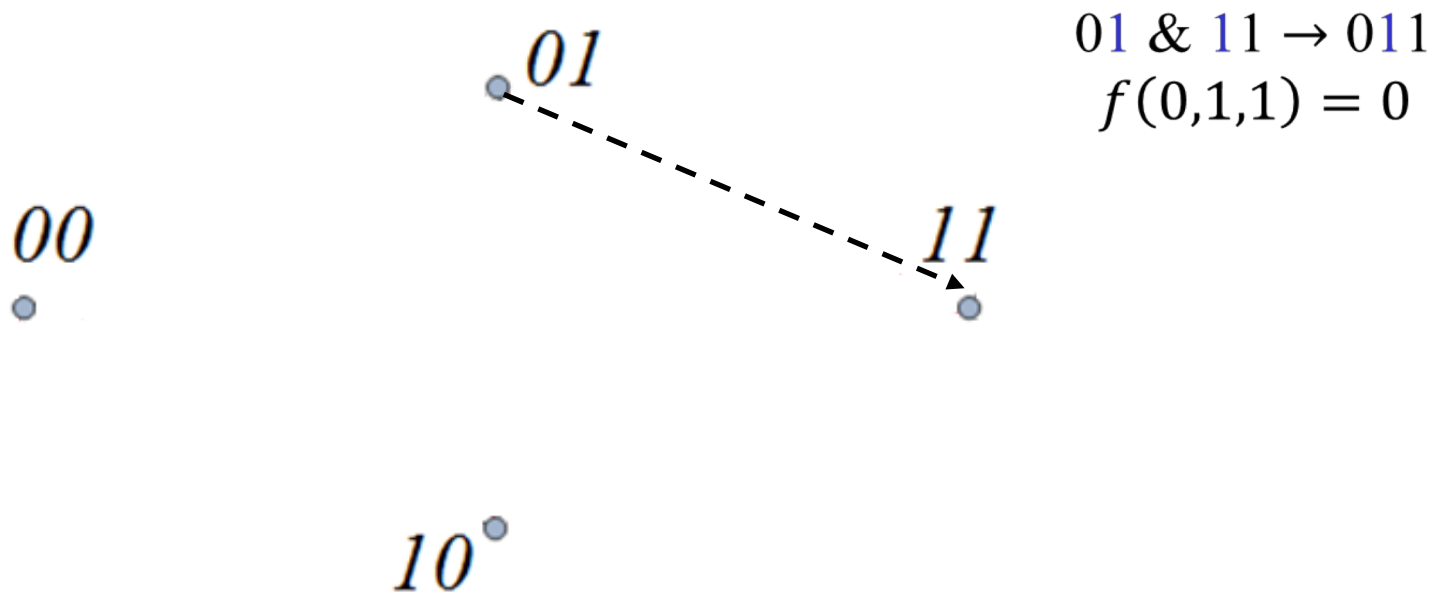
De Bruijn graph

Example (rule 150): $f(x, y, z) = (x + y + z) mod\ 2$.



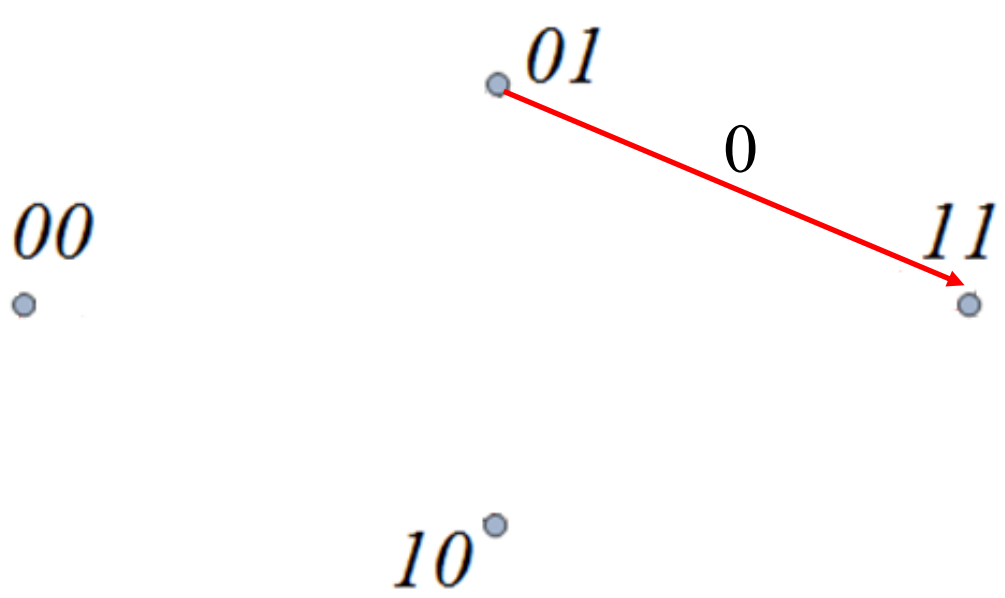
De Bruijn graph

Example (rule 150): $f(x, y, z) = (x + y + z) mod\ 2$.



De Bruijn graph

Example (rule 150): $f(x, y, z) = (x + y + z) mod\ 2$.



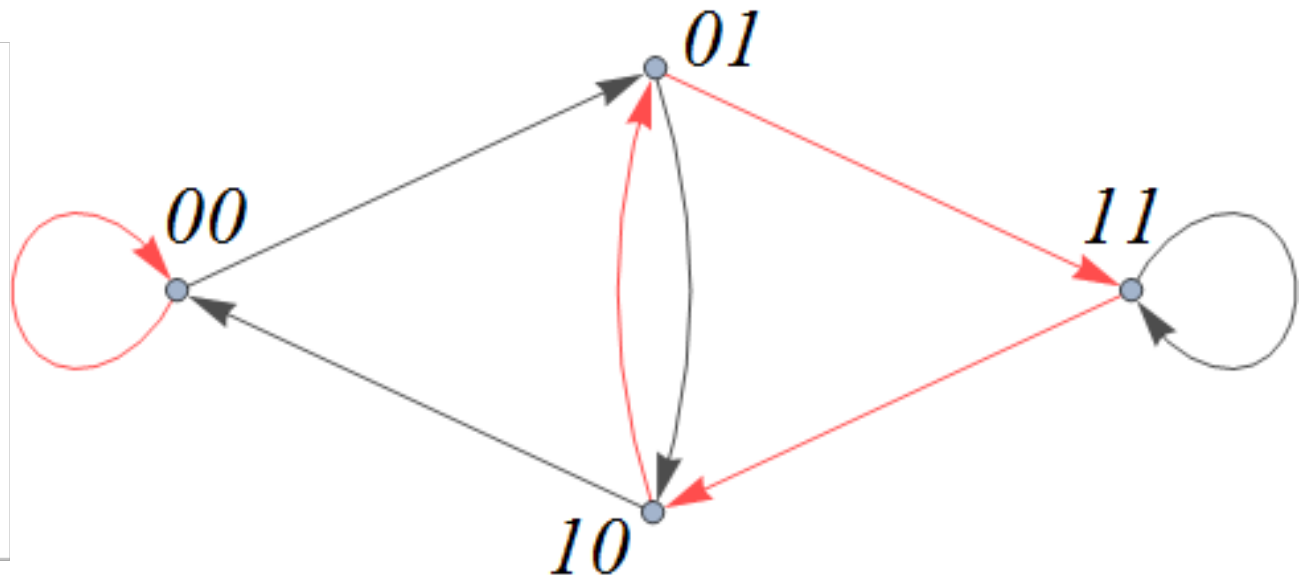
01 & 11 → 011

$f(0,1,1) = 0$

De Bruijn graph

Example (rule 150): $f(x, y, z) = (x + y + z) mod\ 2.$

```
{{1, 1, 1}, 1}
{{1, 1, 0}, 0}
{{1, 0, 1}, 0}
{{1, 0, 0}, 1}
{{0, 1, 1}, 0}
{{0, 1, 0}, 1}
{{0, 0, 1}, 1}
{{0, 0, 0}, 0}
```



PROCESS GRAPHS OF ECAS

De Bruijn graph

- The labels of the graph traversed along a path Correspond to binary sequences that can be present in a configuration in Ω_f^1 .

De Bruijn graph

Example: What is the image of 011001 under Rule 150?

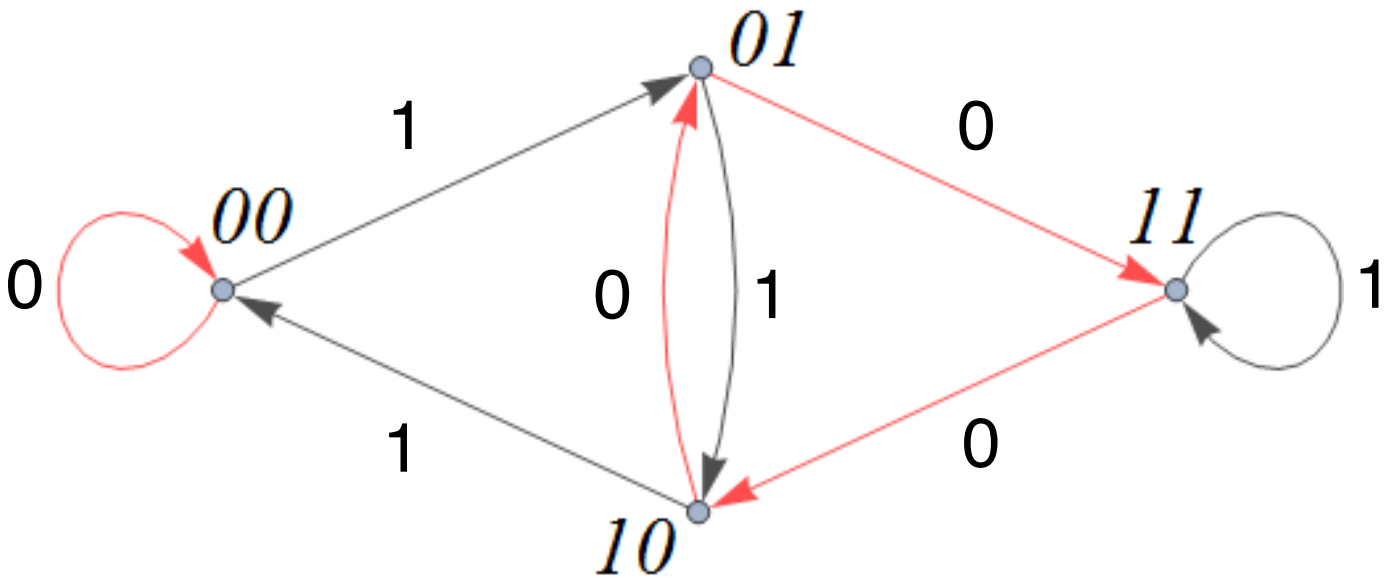
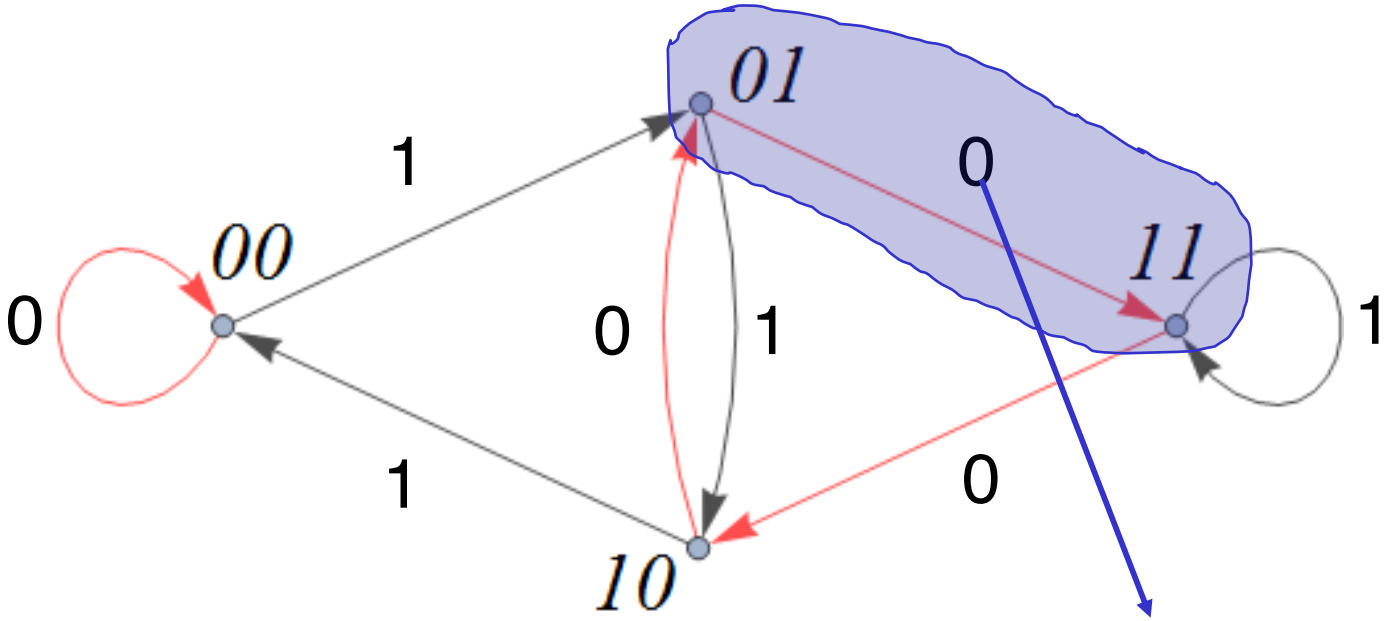


Image:

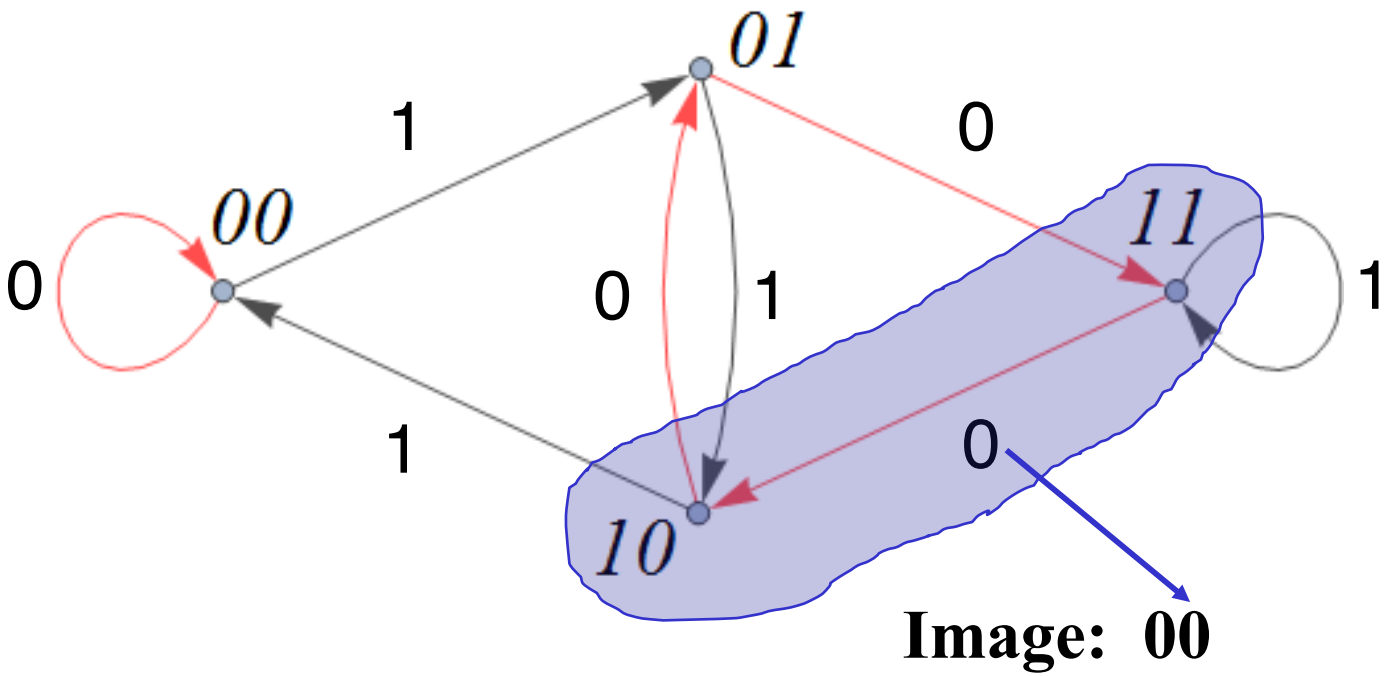
De Bruijn graph

Example: What is the image of 011001 under Rule 150?



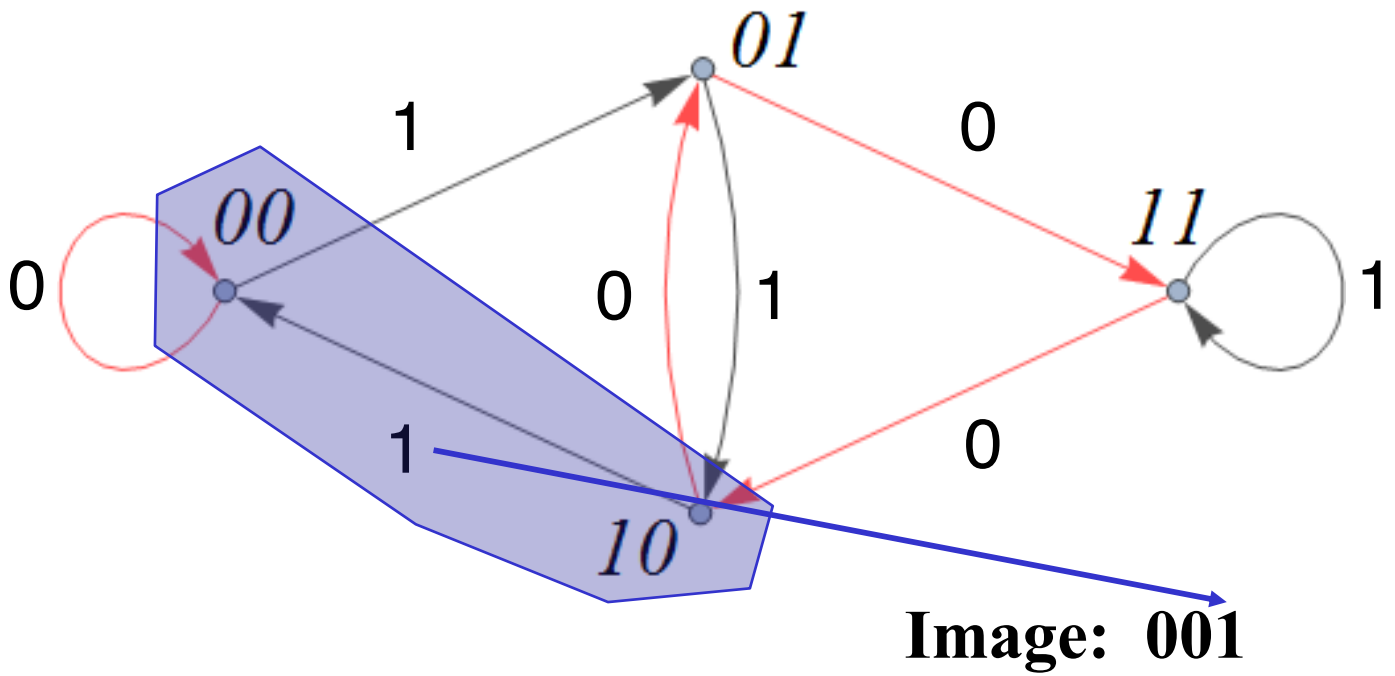
De Bruijn graph

Example: What is the image of 0110101 under Rule 150?



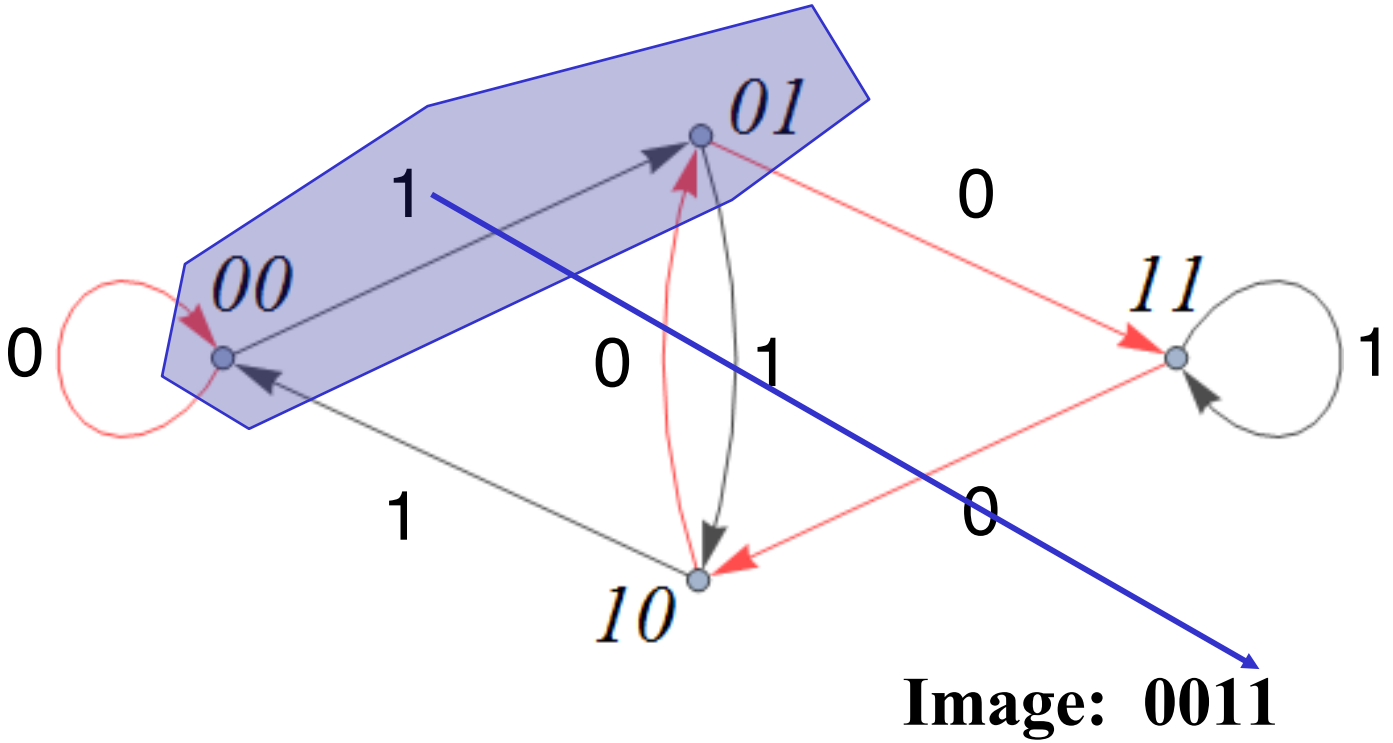
De Bruijn graph

Example: What is the image of 0111001 under Rule 150?



De Bruijn graph

Example: What is the image of 011001 under Rule 150?



De Bruijn graph

Example: What is the image of 011001 under Rule 150?

Answer: 0011

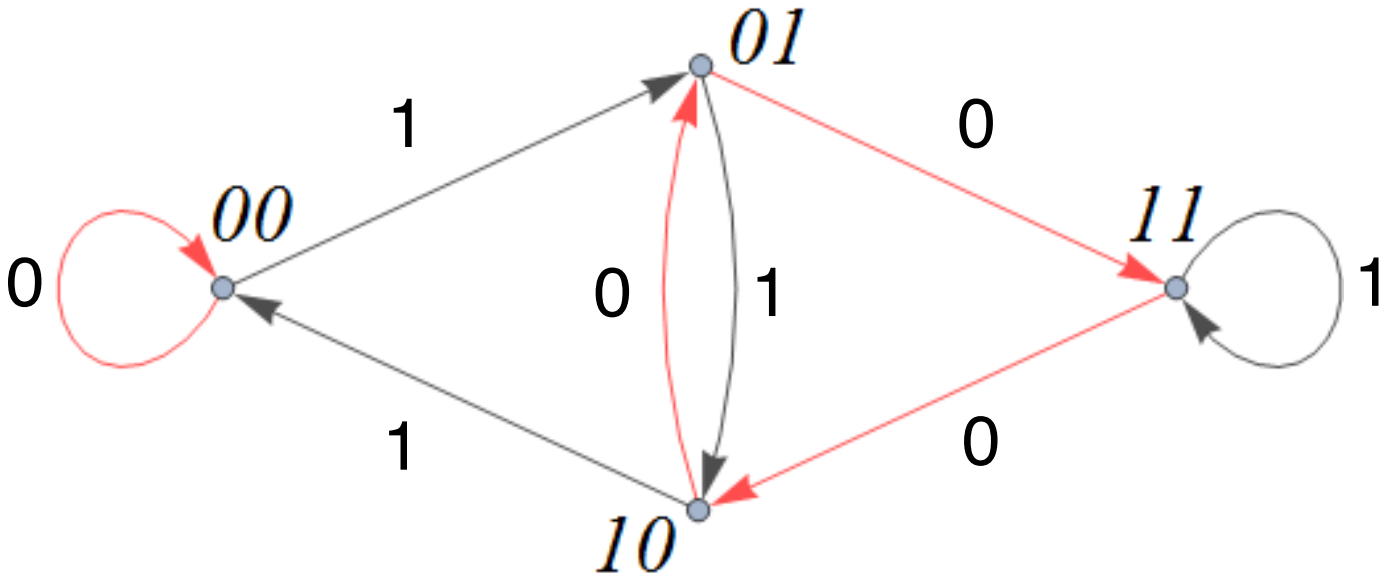


Image: 0011

De Bruijn graph

Question: is it possible to generalise this idea for more iterations? That is, is there an analogous to the De Bruijn graph describing Ω_f^t for $t > 1$?

$$\Omega_f^t = F\left(\Omega_f^{t-1}\right).$$

De Bruijn graph

- Also, the De Bruijn graph tells us if a finite word may be present in a configuration in Ω_f^1 .

De Bruijn graph

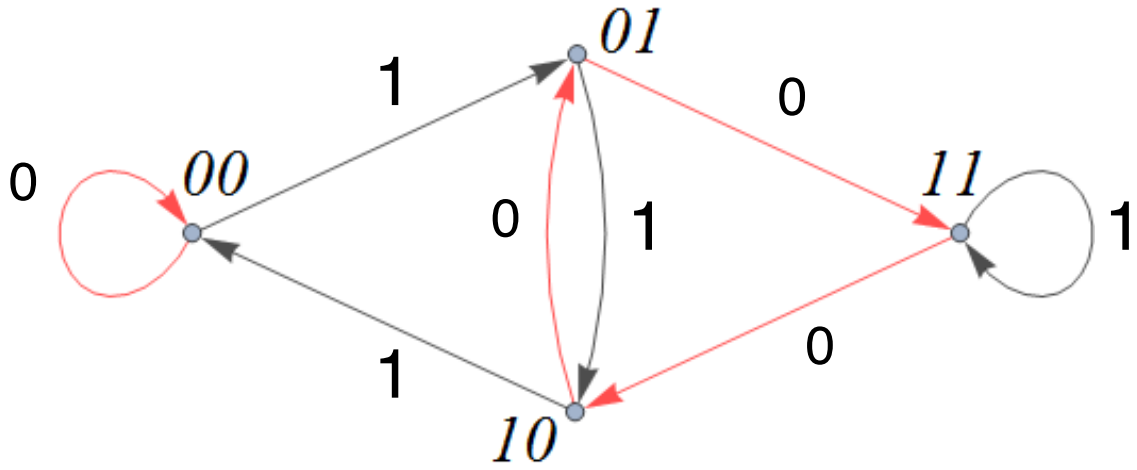
Example: Would the word 00101 be present in Ω_{150}^{1} ?

De Bruijn graph

Example: Would the word 00101 be present in Ω_{150}^{1} ?
Or, rephrasing the question:
Is there a path in the De Bruijn graph whose labels form the word 00101?

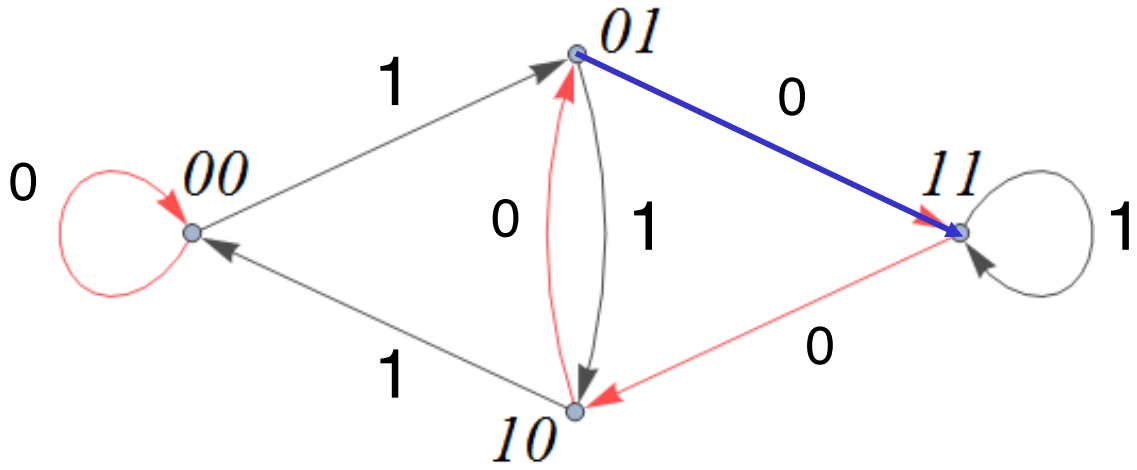
De Bruijn graph

Example: Would the word 00101 be present in Ω_{150}^{1} ?
Or, rephrasing the question:
Is there a path in the De Bruijn graph whose labels form the word 00101?



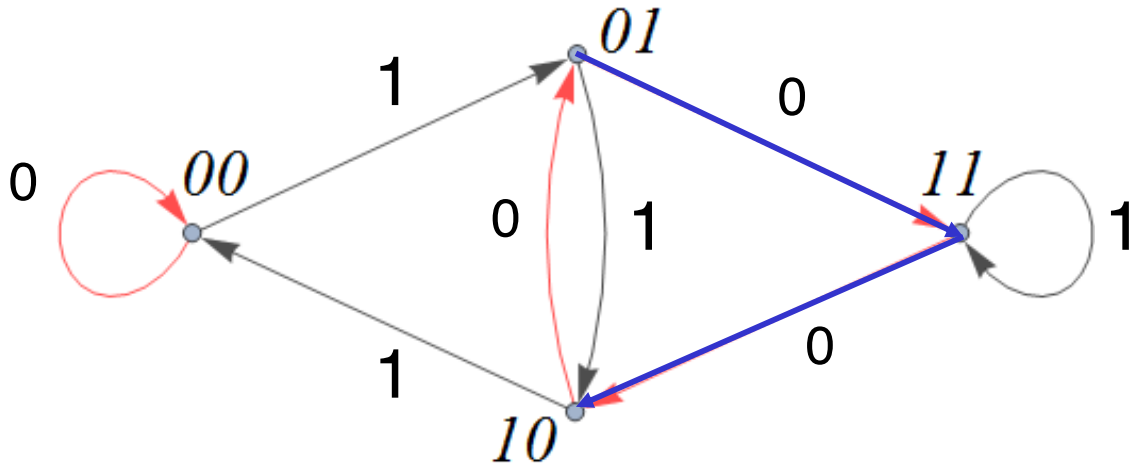
De Bruijn graph

Example: Would the word 00101 be present in Ω^1_{150} ?
Or, rephrasing the question:
Is there a path in the De Bruijn graph whose labels form the word 00101?



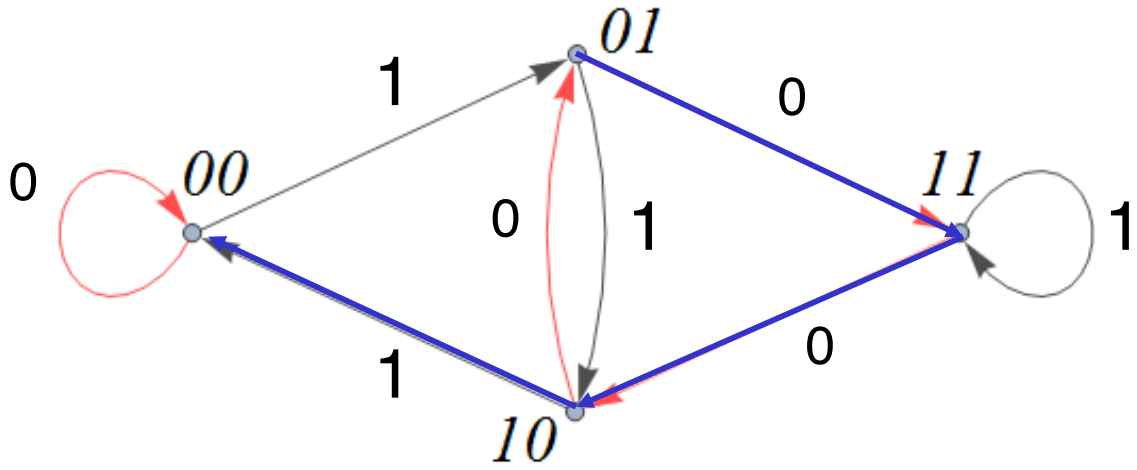
De Bruijn graph

Example: Would the word 00101 be present in Ω_{150}^{1}?
Or, rephrasing the question:
Is there a path in the De Bruijn graph whose labels form the word 00101?



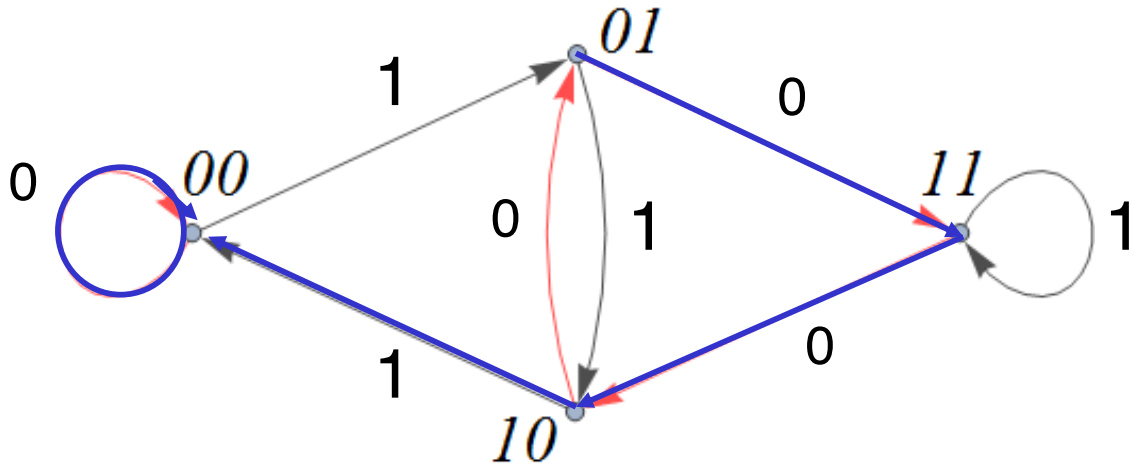
De Bruijn graph

Example: Would the word 00101 be present in Ω_{150}^{1} ?
Or, rephrasing the question:
Is there a path in the De Bruijn graph whose labels form the word 00101?



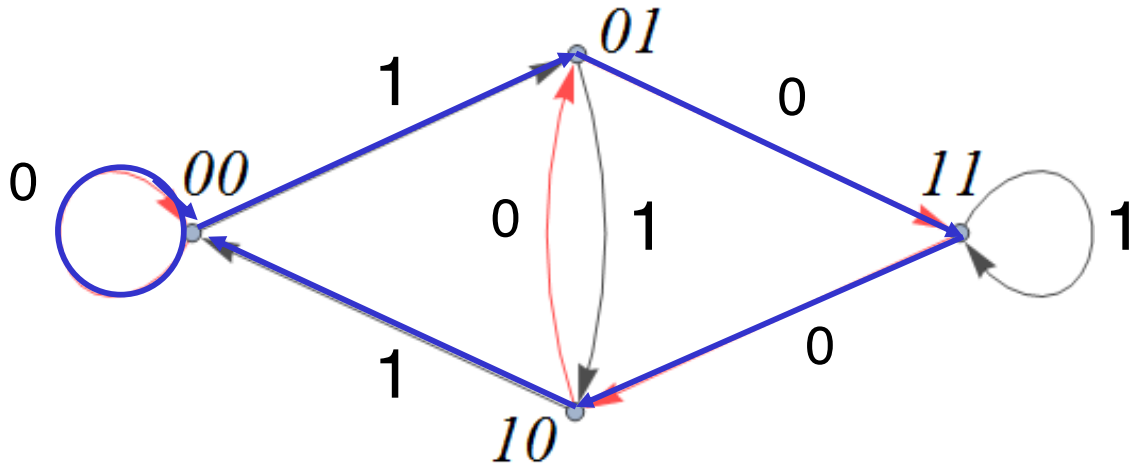
De Bruijn graph

Example: Would the word 00101 be present in Ω_{150}^{1} ?
Or, rephrasing the question:
Is there a path in the De Bruijn graph whose labels form the word 00101?



De Bruijn graph

Example: Would the word 00101 be present in Ω_{150}^{1} ?
Or, rephrasing the question:
Is there a path in the De Bruijn graph whose labels form the word 00101?



De Bruijn graph

- Hence, a configuration is in Ω_{150}^{1} if all of its subwords are "accepted" by the De Bruijn graph.

- De Bruijn graph is a particular kind of **deterministic finite automaton (DFA)**, where all states (nodes) are both initial and final (accepting).

Process graphs

t finite

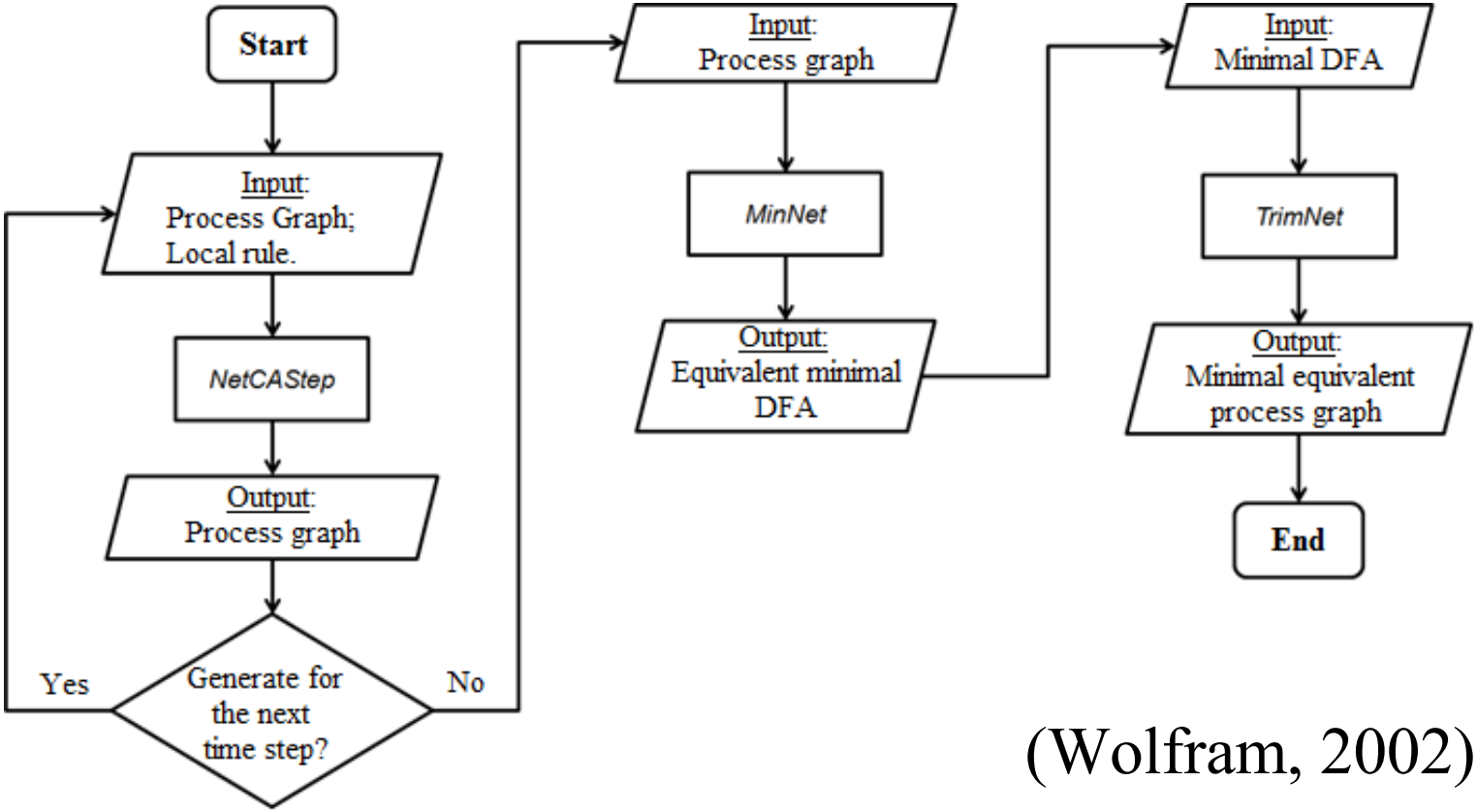
Wolfram (1984,2002) has shown that every set Ω_f^t can be described by a regular language (RL) and be represented by a DFA. In particular, these RLs are factorial, that is, given word in the language, its subwords are also in the language.

Hence, every set Ω_f^t can be described by a factorial RL, named here a *process language*.

Process graphs

Process graphs are the generalisation of
De Bruijn graphs for higher order iterations
of a rule over the set of all binary configurations
and are regarded in the same way.

Process graphs – iterative method



(Wolfram, 2002)

Regarding *TrimNet*

Any process graph can be converted to a (standard) DFA.

But not every DFA can be converted to a process graph, (since not every RL is a factorial RL, i.e., process language).

Reminder: the sets of configurations obtained by iterating an ECA rule **finitely many times** over the set of initial configurations can be described by a factorial RL (Wolfram, 1984).

Regarding *TrimNet*

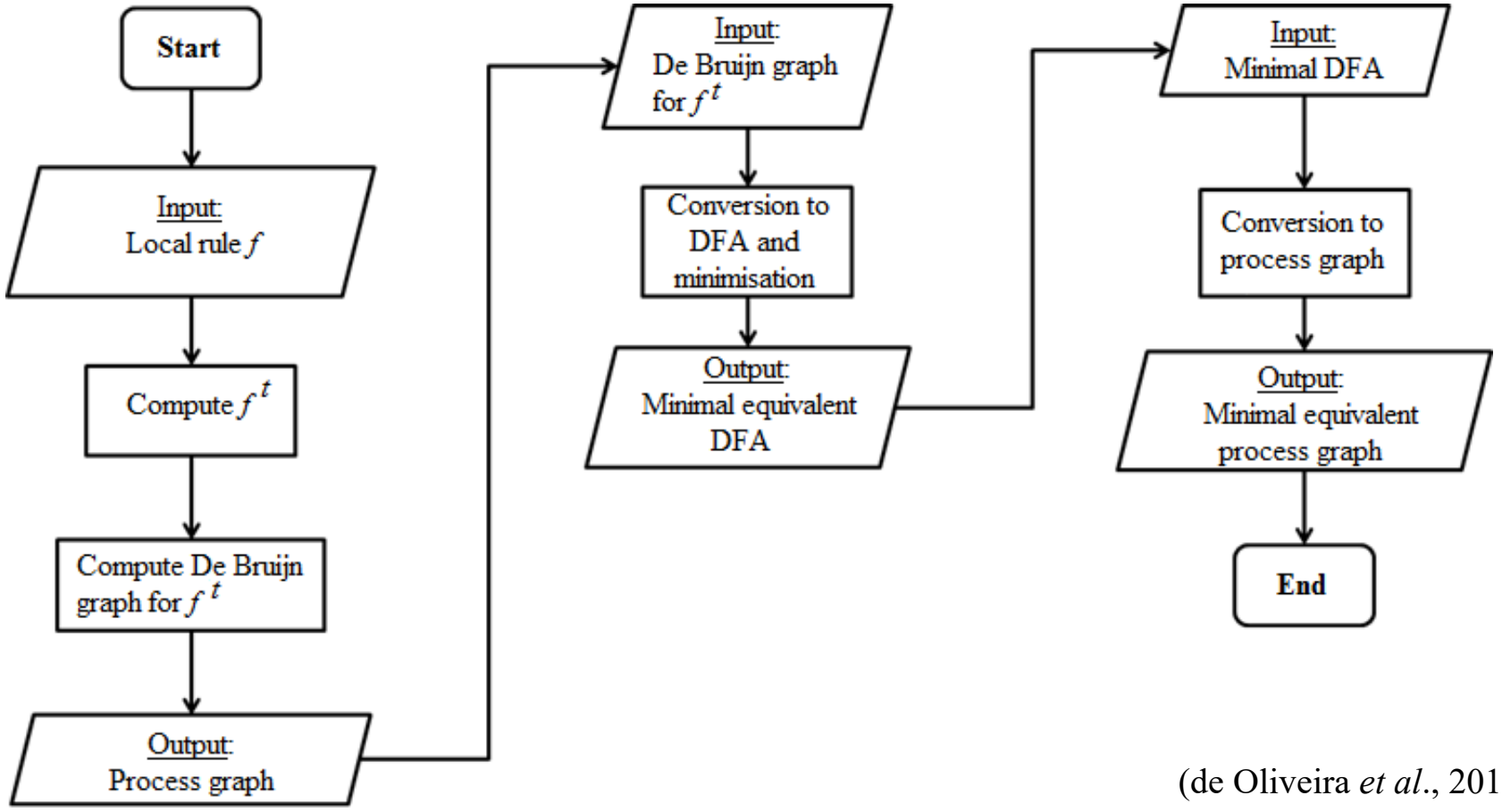
A DFA may only be converted to a process graph if each subpath of a path that leads to an accepting state is itself made of accepting states only.

Hence, what *TrimNet* really does is simply removing the vertices that cannot be reached from the initial state.

Process graphs

NetCAStep may be replaced by a more intuitive procedure: the process graph for time t is exactly the De Bruijn graph for rule f^t.

Process graphs – direct method



(de Oliveira *et al.*, 2015)

Process graphs – direct method

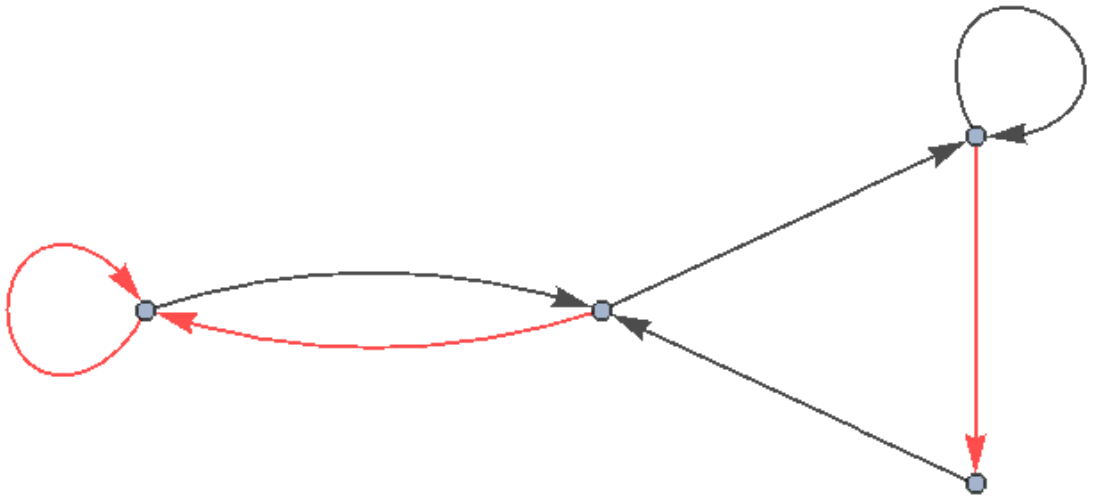
The direct method along with alternative functions for minimising DFAs and for converting them to process graphs made possible for us to to compute graphs describing the ECA rule space which were not previously present in the literature.

(Wolfram's *Regular Language Complexity* table)

2. Growth of the ECAs' process graphs

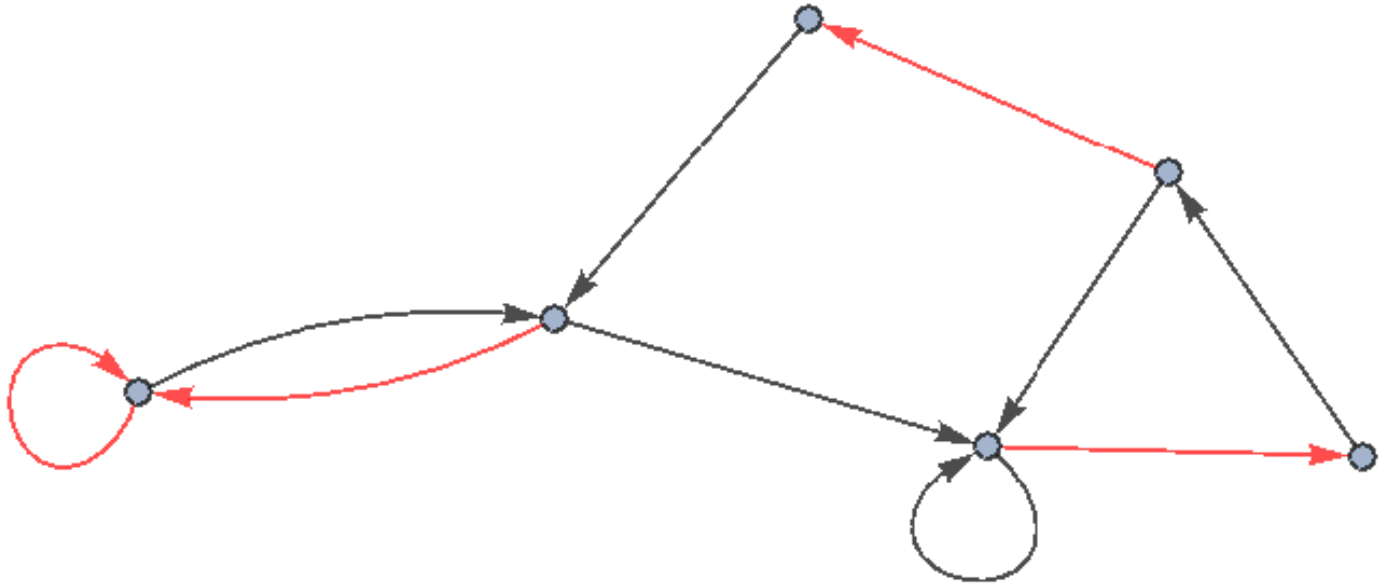
Process graphs

Example: process graphs of rule 184
$t = 1$



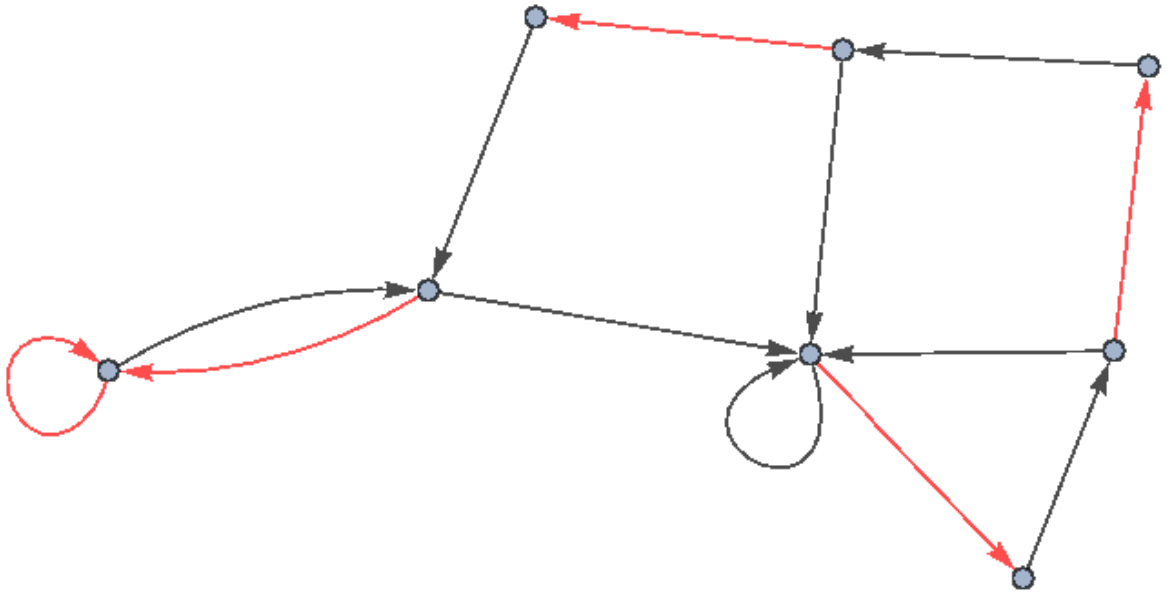
Process graphs

Example: process graphs of rule 184
$t = 2$



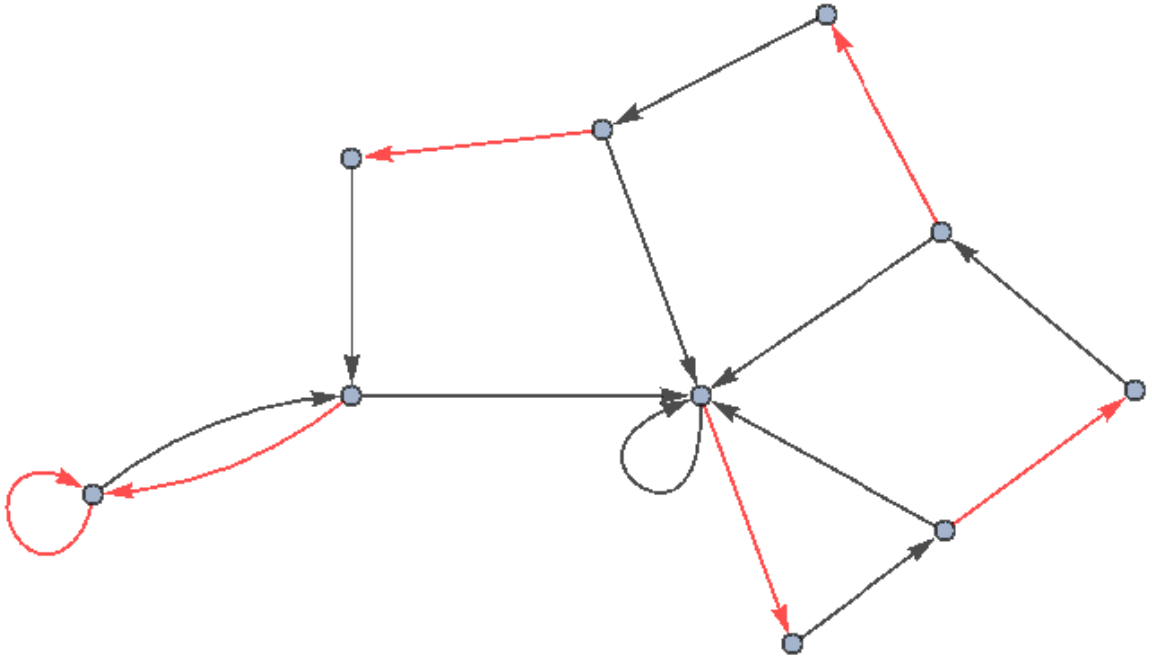
Process graphs

Example: process graphs of rule 184
$t = 3$



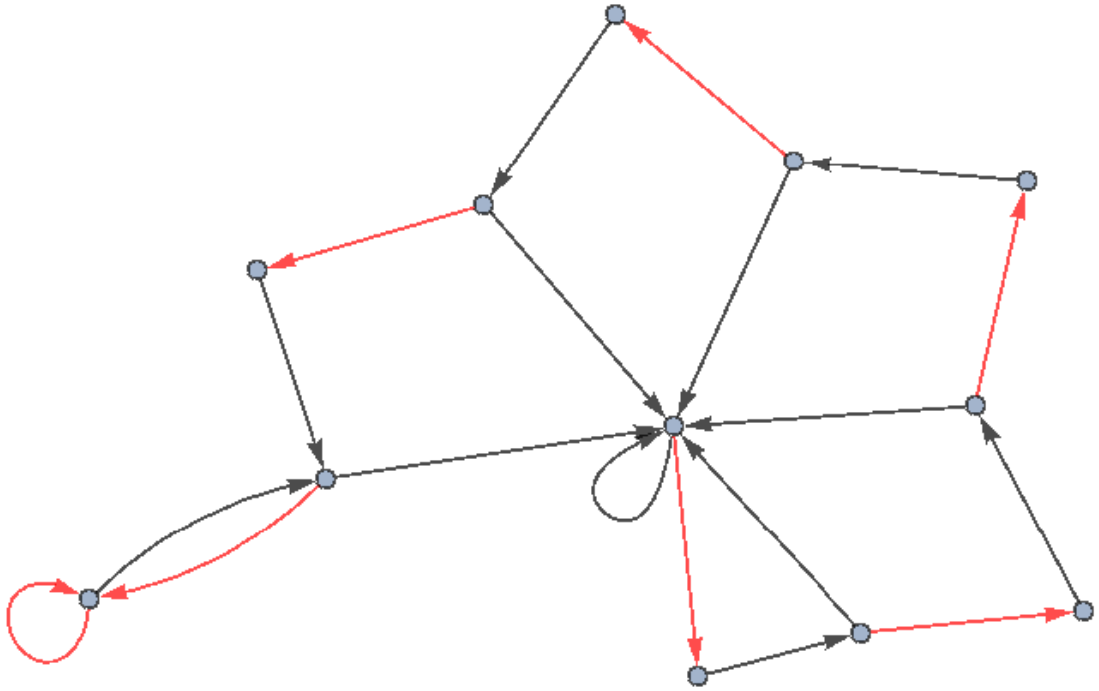
Process graphs

Example: process graphs of rule 184
$t = 4$



Process graphs

Example: process graphs of rule 184
$t = 5$



GROWTH OF ECAS PROCESS GRAPHS

Process graphs

Question: Is there a pattern in the growth of these graphs? And, if that is the case, how is it possible to compute the pattern?

INFERENCE OF LIMIT GRAPHS

Our original approach towards the limit graphs:

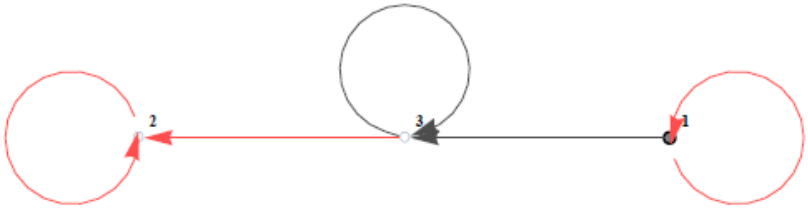
Some limit graphs were inferred by **visual analysis** the growth of the graphs.

(The following limit graphs, obtained by visual inspection of the graphs sequences are the same obtained **later**, by looking at the evolution of their corresponding **regular expressions**.)

INFERENCE OF LIMIT GRAPHS

Rule 32

Rule 128 & Rule 136

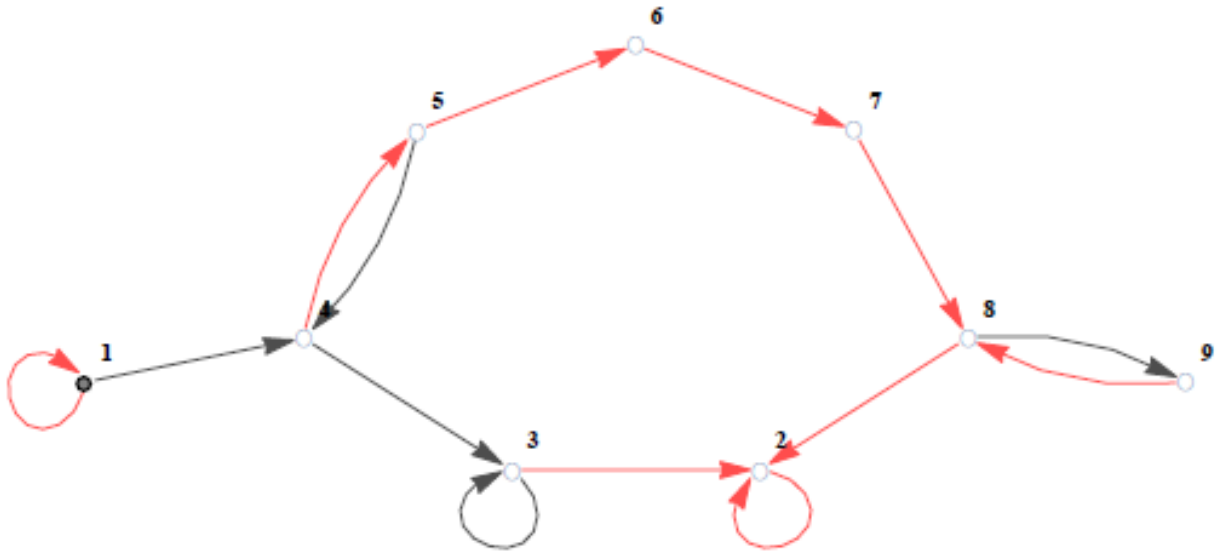


Rule 168

Rule 224

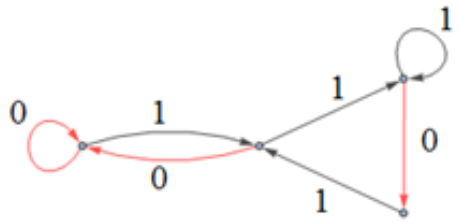
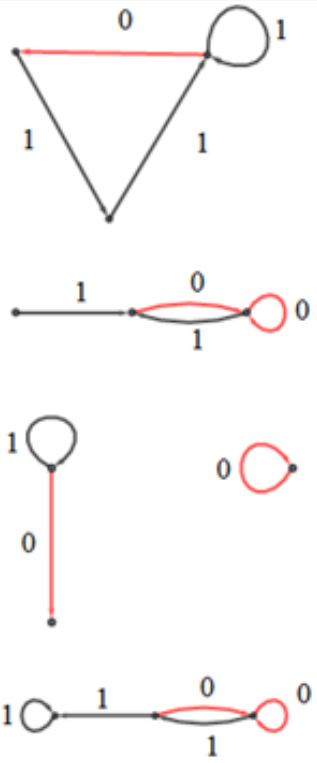
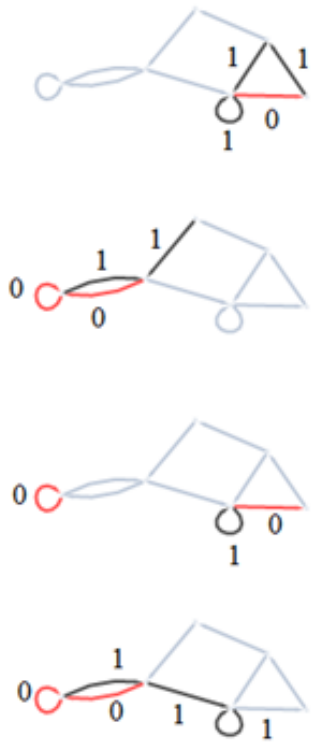
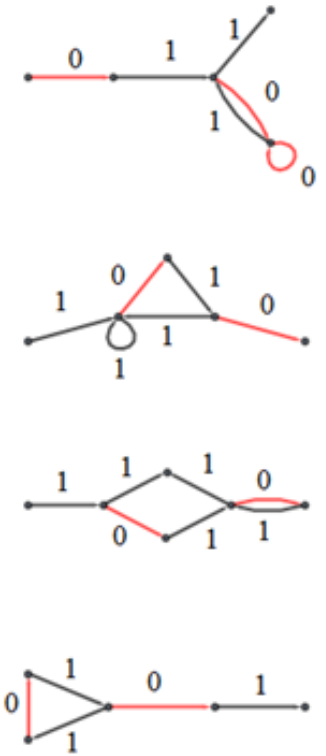
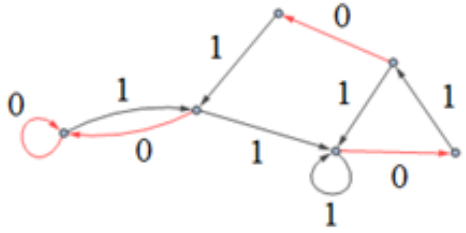
INFERENCE OF LIMIT GRAPHS

Rule 160
(visual inference only, not by analysis of the corresponding regular expressions.

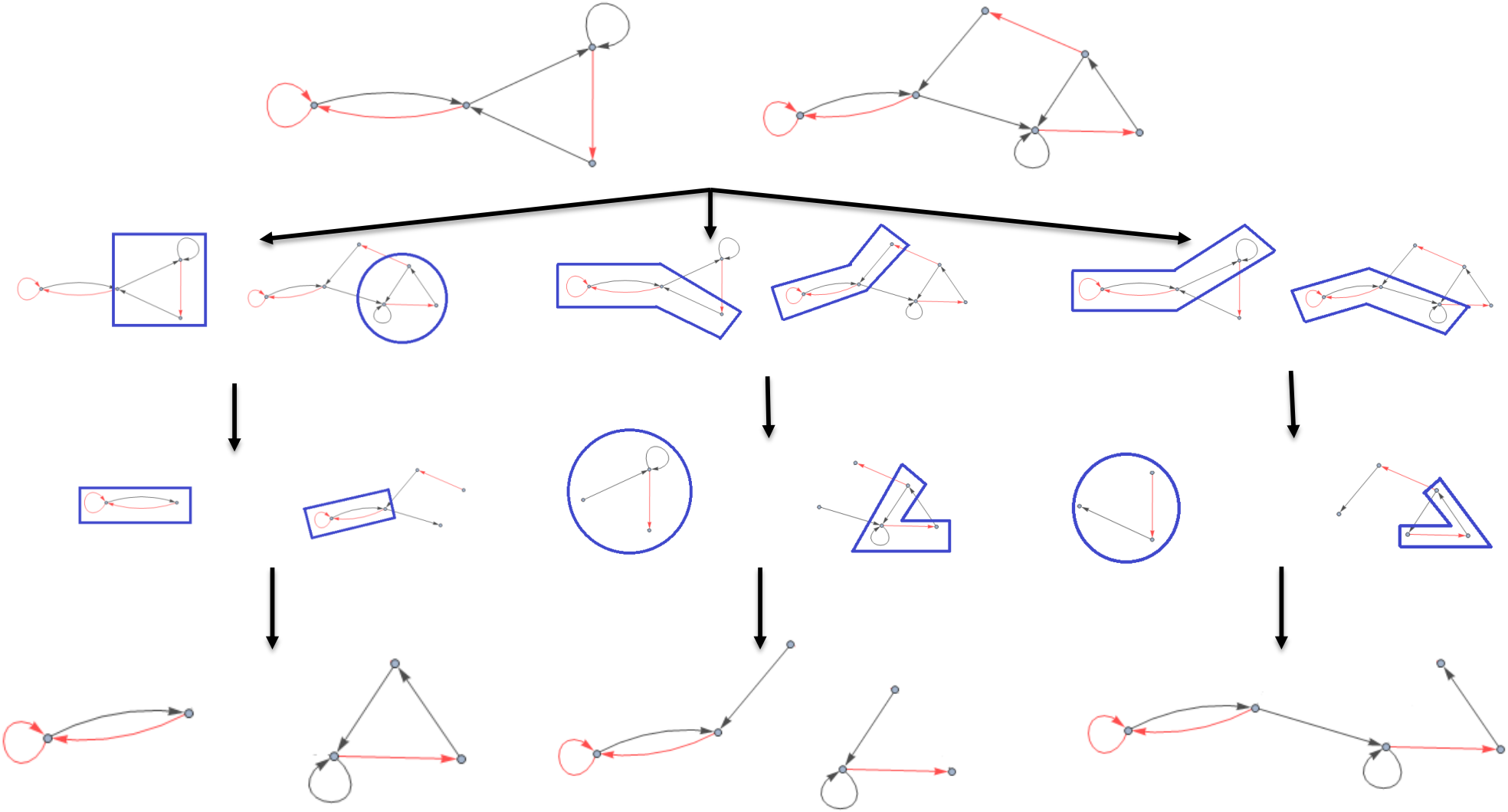


3. Covers and difference sets

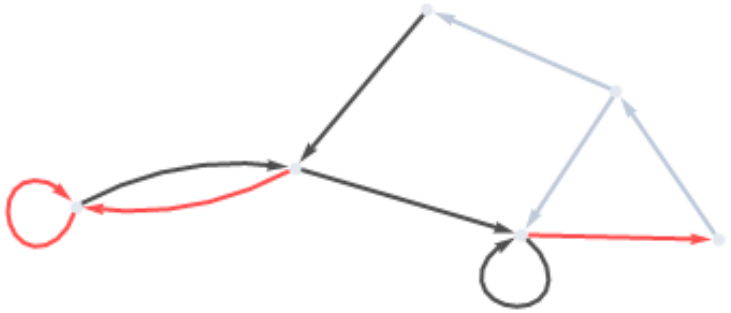
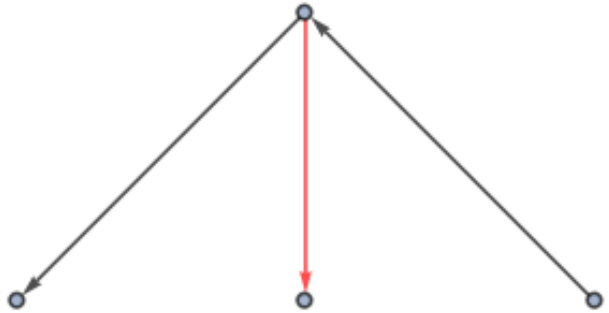
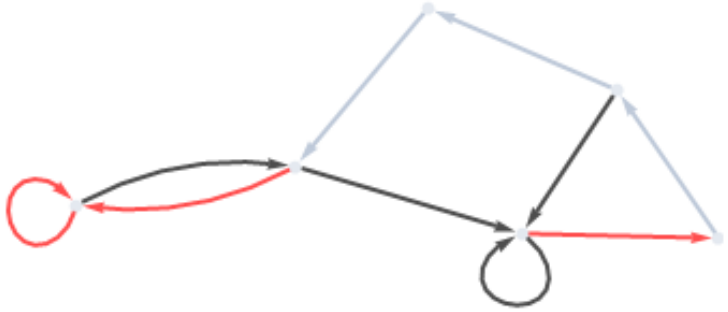
Complement graph

Graph - Rule 184 (t = 1)	*Isomorphic subgraph*	$G(E_\phi)$	$G(E_\phi)^C$
			
Graph - Rule 184 (t = 2)			
			

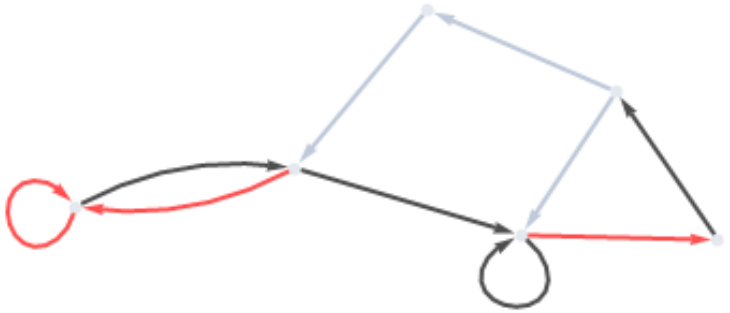
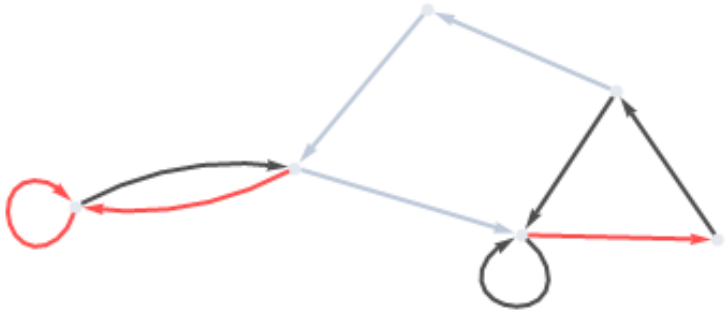
Covers and Difference Sets



Covers and differences

Cover	*Difference*
	
	

Covers and differences

Cover	*Difference*
	
	

Study of the process graphs growth patterns

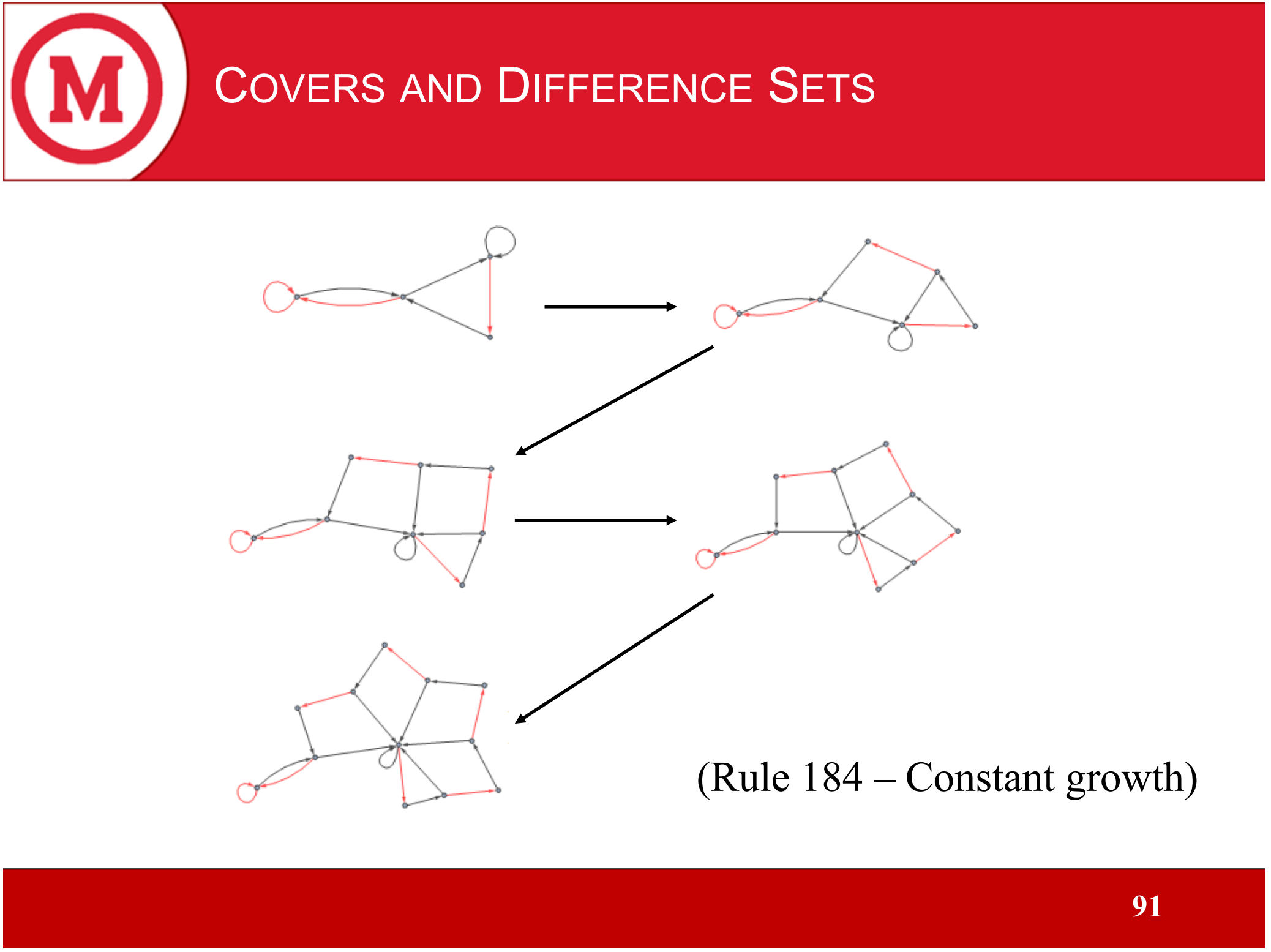
Type	Rules	Amount
Trivial	0, 1, 2, 3, 4, 5, 8, 10, 12, 15, 19, 24, 29, 30, 34, 36, 38, 42, 45, 46, 51, 60, 72, 76, 90, 105, 106, 108, 138, 150, 154, 170, 200, 204	34

Study of the process graphs growth patterns

Type	Rules	Amount
Not analysed (due to computational limitations)	6, 9, 18, 22, 25, 26, 37, 41, 54, 58, 62, 73, 74, 94, 104, 110, 122, 126, 134, 146, 156, 160, 164	23

Study of the process graphs growth patterns

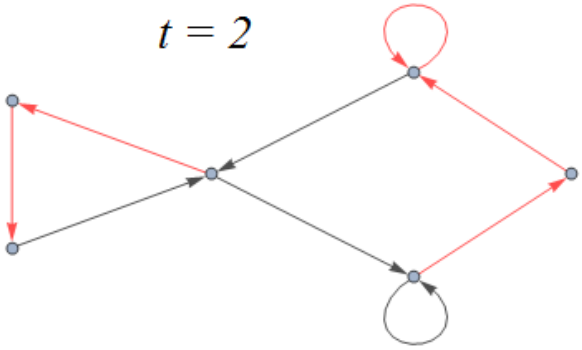
Type	Rules	Amount
Constant	23, 32, 40, 43, 56, 128, 130, 132, 136, 140, 142, 162, 168, 172, 184, 232	16

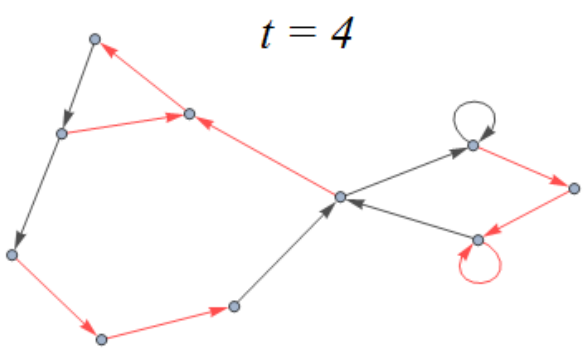

COVERS AND DIFFERENCE SETS
(Rule 184 – Constant growth)

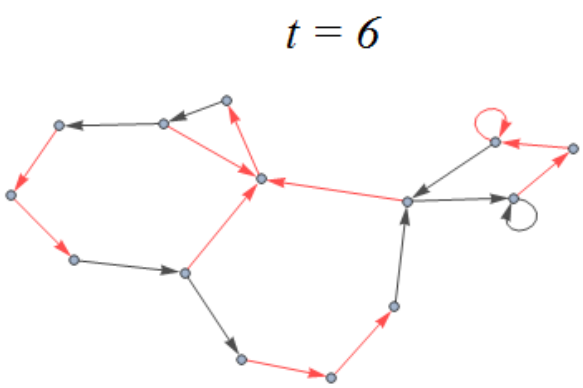
Study of the process graphs growth patterns

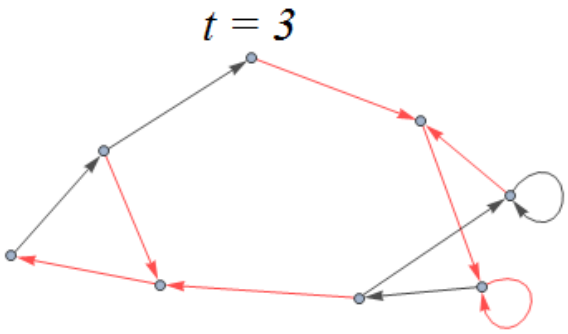
Type	Rules	Amount
Periodically constant	7, 11, 13, 14, 35, 50	6

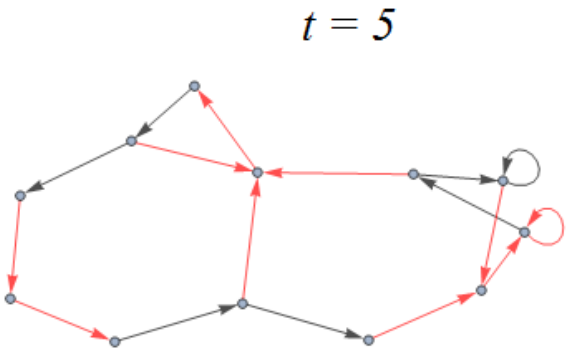
COVERS AND DIFFERENCE SETS

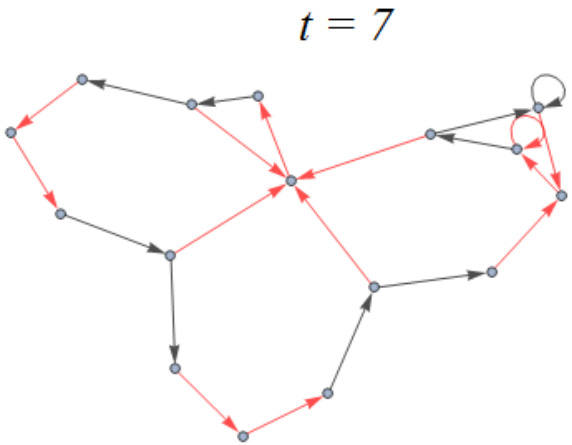










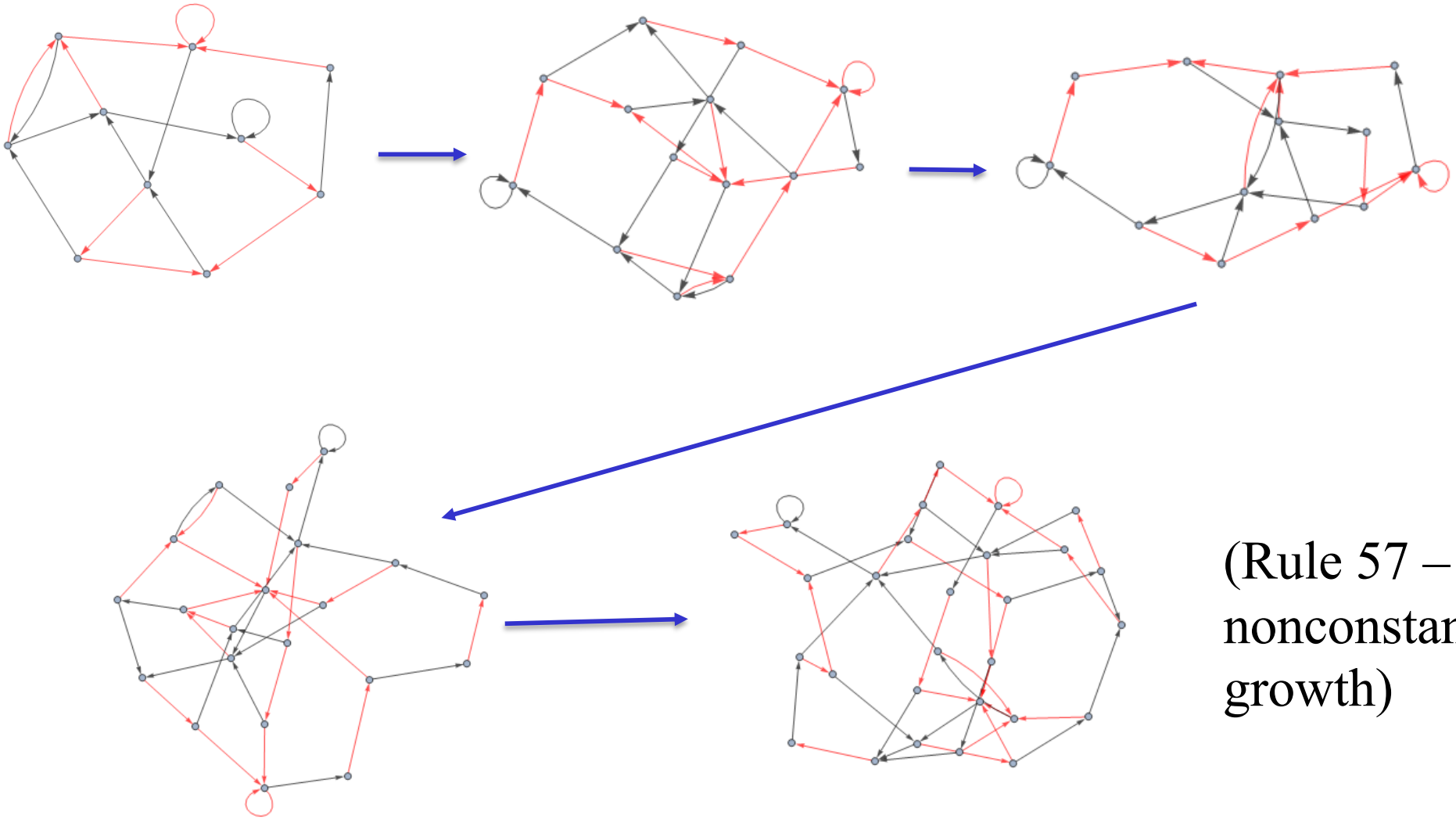


(Rule 11 – Periodically constant growth)

Study of the process graphs growth patterns

Type	Rules	Amount
Nonconstant	27, 28, 33, 44, 57 77, 78, 152, 178	9

Covers and Difference Sets



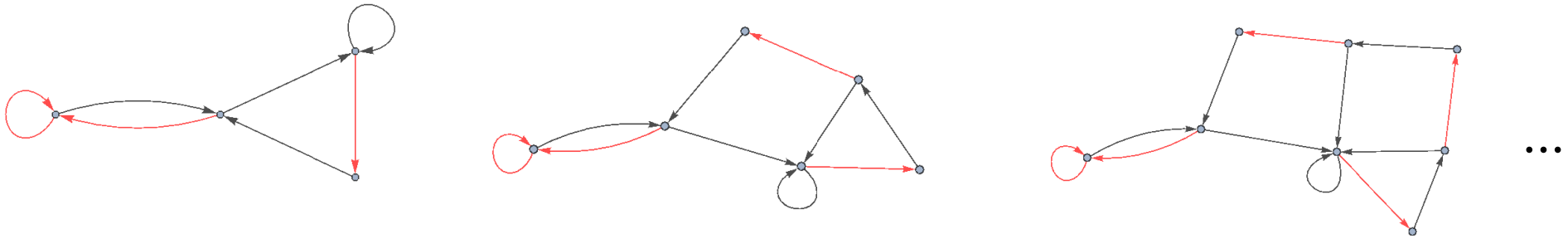
(Rule 57 – nonconstant growth)

Inference of finite-time process graphs

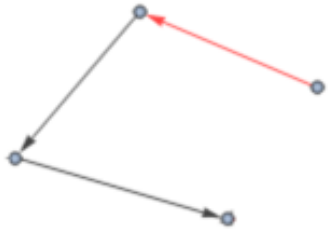
For the rules with constant or periodically constant growth patterns, it is possible to infer process graphs for larger time steps without computing them (by the iterative or the direct methods) based upon the added structure (difference) from one graph to another.

Inference of finite-time process graphs

Example: Rule 184

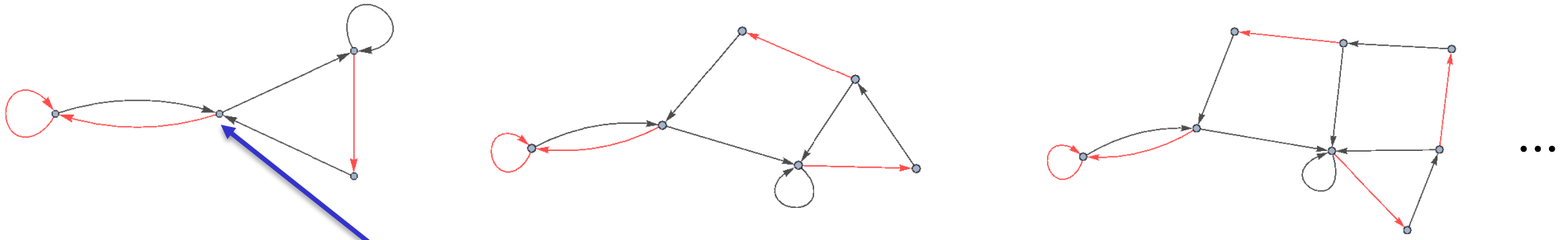


Added structure (difference)

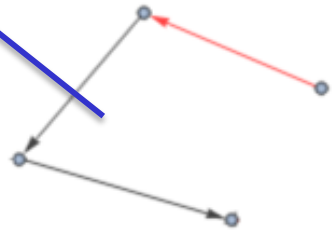


Inference of finite-time process graphs

Example: Rule 184

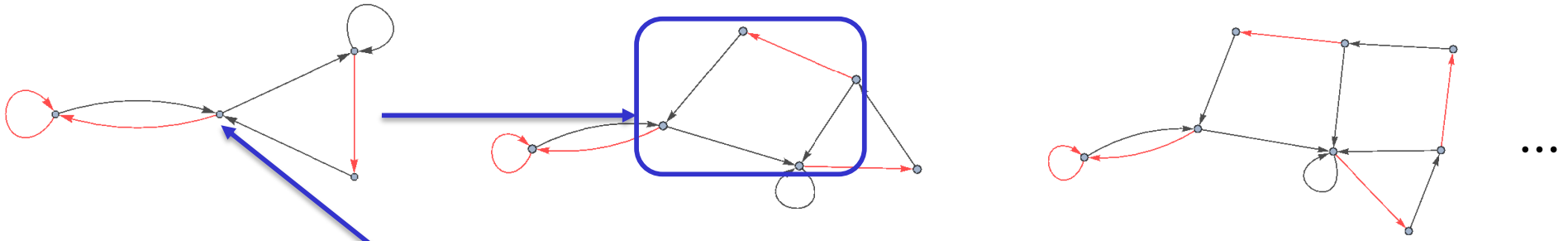


Added structure (difference)

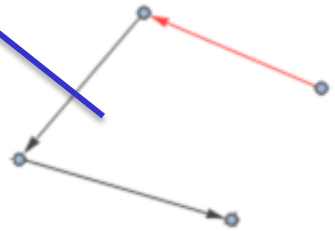


Inference of finite-time process graphs

Example: Rule 184

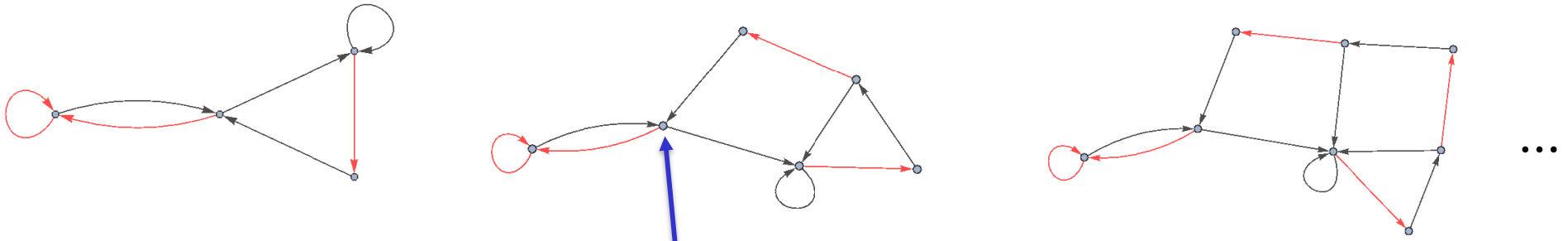


Added structure (difference)

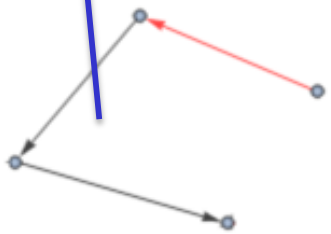


Inference of finite-time process graphs

Example: Rule 184

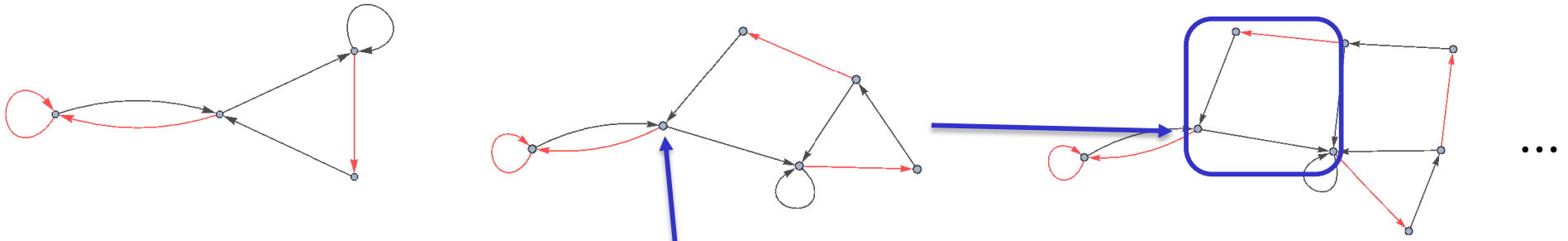


Added structure (difference)

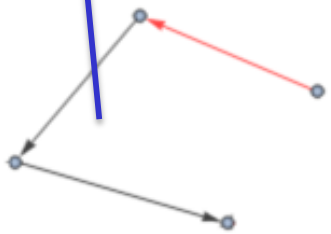


Inference of finite-time process graphs

Example: Rule 184

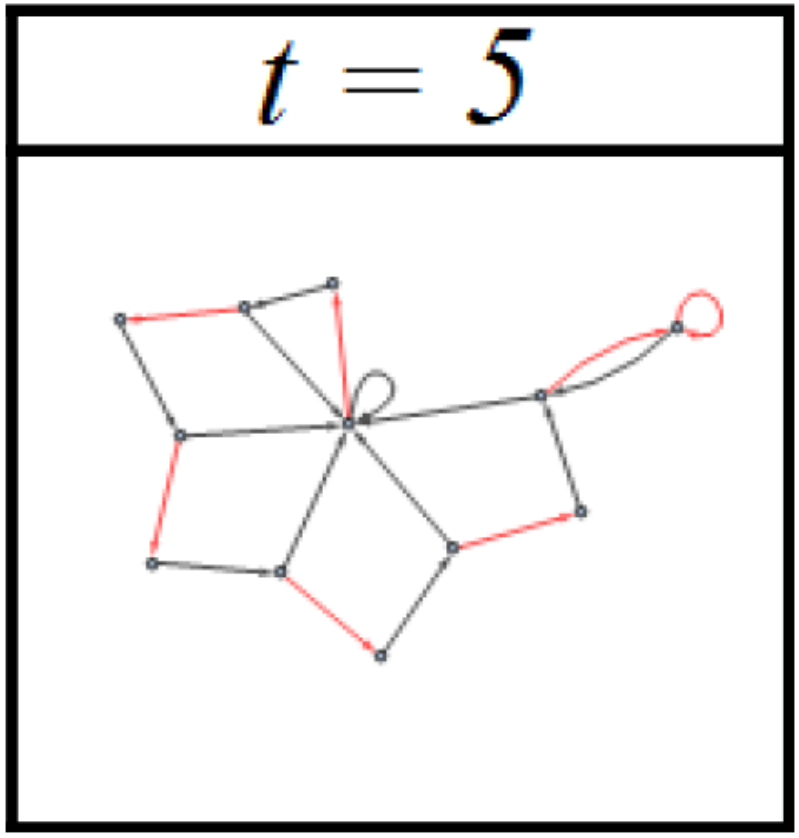


Added structure (difference)



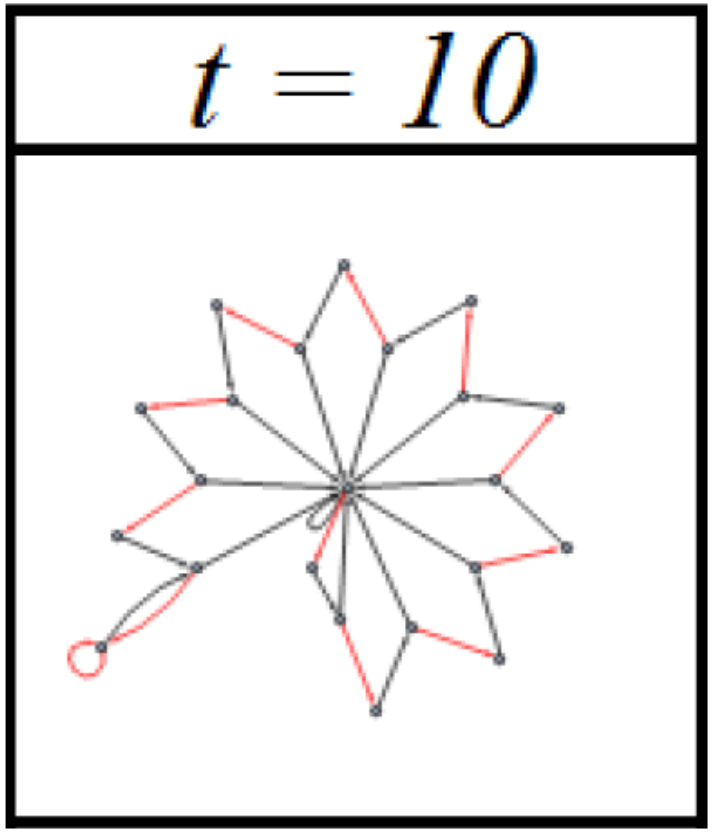
Inference of finite-time process graphs

Rule 184



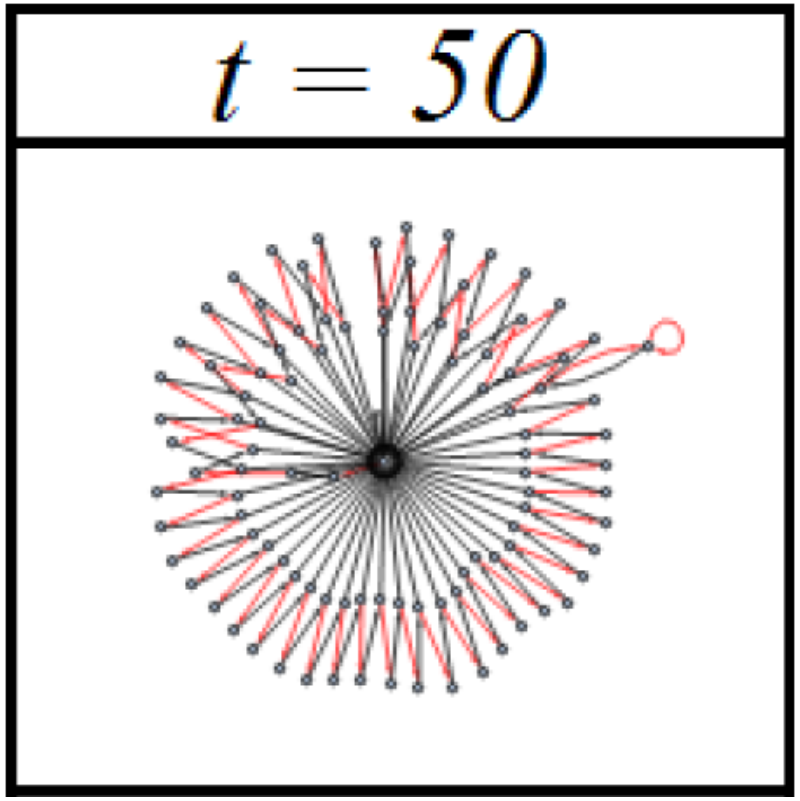
Inference of finite-time process graphs

Rule 184



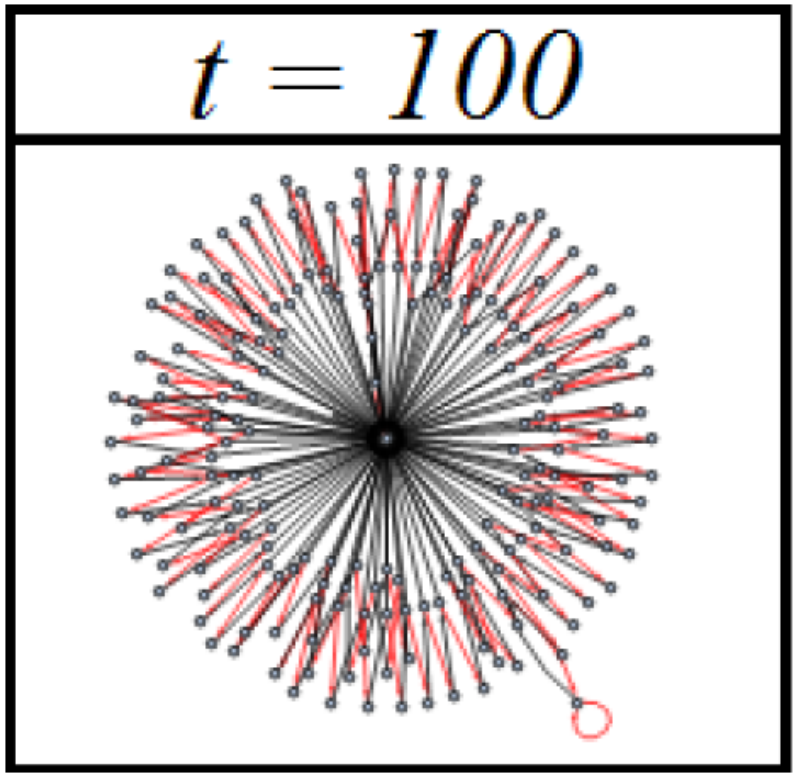
Inference of finite-time process graphs

Rule 184



Inference of finite-time process graphs

Rule 184



Inference of finite-time process graphs

Problem: Obtaining finite-time process graphs is possible by analysing the difference sets, but obtaining a **limit graph** is not trivial.

What may be used to infer the limit graphs?

4. Inference of limit graphs

Analysis of regular expressions

Solution: Dual approach: to **look at the regular expressions** describing the languages accepted by the process graphs.

Generally, finding the limit language of a CA by analytical means is not trivial at all.

Previous findings on the limit languages of some ECA rules:

- ECA 18 is context-sensitive (Jiang & Xie, 2001)
- ECA 146 is context-sensitive (Wang & Morita, 2006)
- ECAs 22 & 122 are not regular (Jiang & Yi, 2005)
- If the limit language of ECA 18 is not regular, then the same happens to the limit languages of ECAs 126 and 146 (Jiang, 2006)
  => therefore 126 is context-sensitive.
- ECA 56 has been analytically given in (Qin & Xie, 2005)
  (the same one we found by our method :-)

Evolution of the regular expressions

Rule 32	
t	Regular expression
1	(0+10)*(ε+1)
2	(0+10(10)*000)*(ε+1+10((10)*(ε+0+1+00)))
3	(0+10(10)*00000)*(ε+1+10((10)*(ε+0+00+000+1+0000)))
4	(0+10(10)*0000000)*(ε+1+10((10)*(ε+0+00+000+0000+00000+1+000000)))
5	(0+10(10)*000000000)*(ε+1+10((10)*(ε+0+00+000+0000+00000+000000+0000000+1+00000000)))

$$((0+10(10)*0^{2^{n-1}})*(\varepsilon+1+10((10)*(\varepsilon+1+\sum_{i=1}^{2^{n-1}}0^i))))$$

INFERENCE OF LIMIT GRAPHS

Example: Rule 32 (t = 1 to t = 5)

$$((0+10(10)*0^{2^{n-1}})*(\varepsilon+1+10((10)*(\varepsilon+1+\sum_{i=1}^{2^{n-1}}0^i))))$$

$$(0*(\varepsilon+1+10((10)*(\varepsilon+1+00*))))$$

INFERENCE OF LIMIT GRAPHS

Example: Rule 184(t = 1 to t = 3)

$$((0+10+11((1+011)*010))*(\varepsilon+1+11((1+011)*(\varepsilon+0+01))))$$

$$((0+10+11((1+011+01011)*01010))*(\varepsilon+1+11((1+011+01011)*(\varepsilon+0+01(\varepsilon+0)+0101))))$$

$$((0+10+11((1+011+01011+0101011)*0101010))*(\varepsilon+1+11((1+011+01011+0101011)*(\varepsilon+0+01(\varepsilon+0)+0101(\varepsilon+0)+010101))))$$

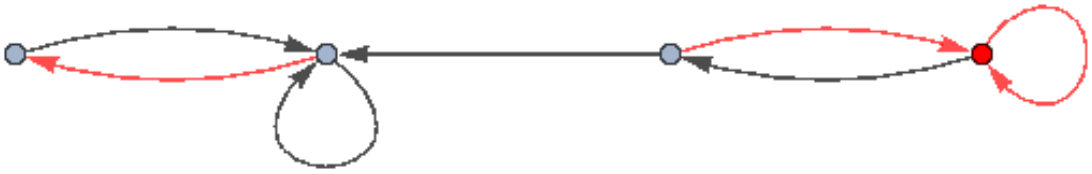
$$((0+10+11((1+0(\sum_{i=0}^{n-1}(10)^i)11)*0(10)^n))*(\varepsilon+1+11((1+0(\sum_{i=0}^{n-1}(10)^i)11)*(\varepsilon+0+(01)^n+(\sum_{i=0}^{n-1}(01)^i)(\varepsilon+0)))))$$

INFERENCE OF LIMIT GRAPHS

Example: Rule 184 (t = 1 to t = 3)

$$((0+10+11((1+0(\sum_{i=0}^{n-1}(10)^i)11)*0(10)^n))*(\varepsilon+1+11((1+0(\sum_{i=0}^{n-1}(10)^i)11)*(\varepsilon+0+(01)^n+(\sum_{i=0}^{n-1}(01)^i)(\varepsilon+0)))))$$

$$((0+10)*(\varepsilon+1+11((1+0(10)*11)*(\varepsilon+0+(01)*(\varepsilon+0)))))$$



INFERENCE OF LIMIT GRAPHS

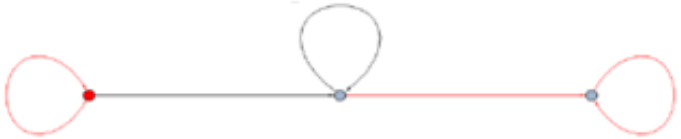
Rule 32

$0^*(\epsilon + 1 + 10((10)^*(0^* + 1))$

Rule 128

$0^*(\epsilon + 1(1^*(\epsilon + 0^+)))$

Rule 136



$0^*(\epsilon + 1(1^*(\epsilon + 0^+)))$

Rule 56

$(0 + 10)^*(\epsilon + 1(\epsilon + 1(\epsilon + 0)) + 1101(((01)^*101)^*$
$((01)^*(\epsilon + 0 + 1(\epsilon + 0)) + (01)^*(\epsilon + 1(\epsilon + 0)))))$

Rule 132

$0^*(\epsilon + 1 + 11(1^*0^*) + 10((0^*10)^*(0^*(\epsilon + 1))))$

INFERENCE OF LIMIT GRAPHS

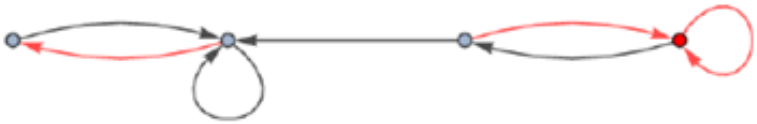
Rule 140

$(0+10)^*(\epsilon+1+11(1^*(\epsilon+0^+)))$

Rule 168

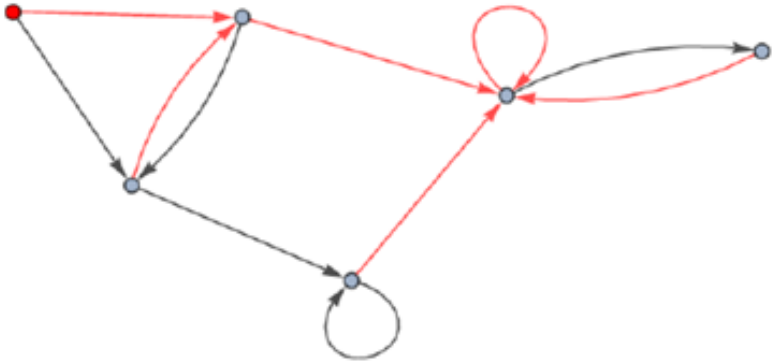
$0^*(\epsilon+1((1+01)^*(\epsilon+0(\epsilon+0^+))))$

Rule 184



$(0+10)^*(\epsilon+1+11((1+0(10)^*11)^*(\epsilon+0+(01)^*(\epsilon+0))))$

Rule 162



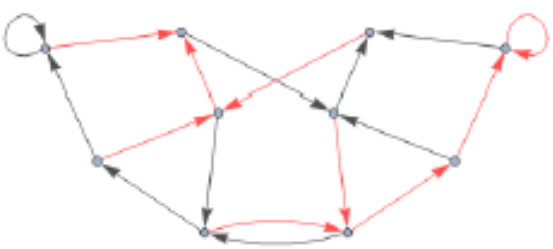
$(\epsilon+0)+00((0+(10)^*100)^*(10)^*(\epsilon+1))+$
$((1+01))((01)^*(\epsilon+0+00((0+(10)^*100)^*(10)^*(\epsilon+$
$1))+1(1^*(\epsilon+0((0+(10)^*100)^*(10)^*(\epsilon+1))))))$

INFERENCE OF LIMIT GRAPHS

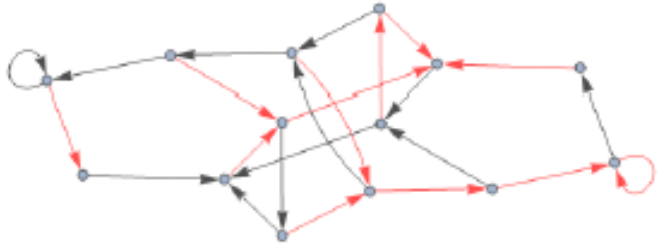
Another possibility is to analyse
subsets of the initial configuration space and
analyse the evolution of the regular expressions that arise from such restriction.

INFERENCE OF LIMIT GRAPHS

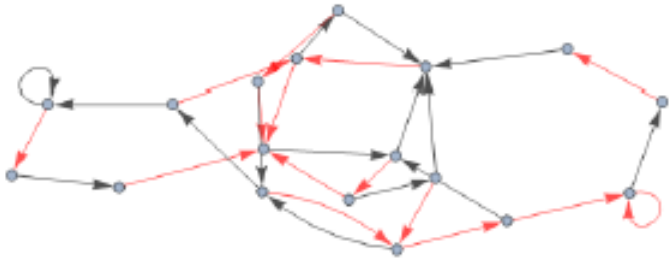
Example: Rule 178



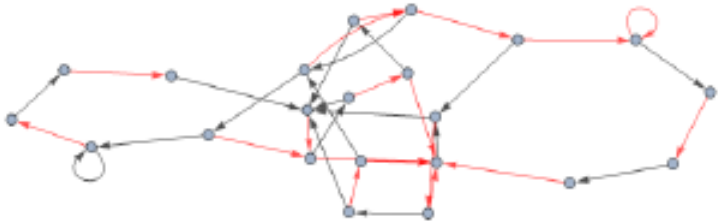
$t = 1$



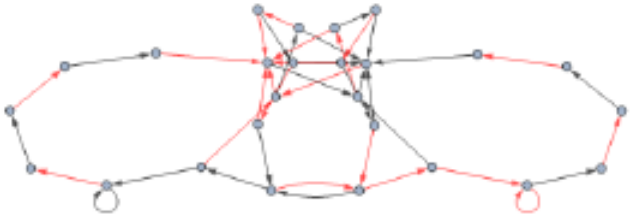
$t = 2$



$t = 3$



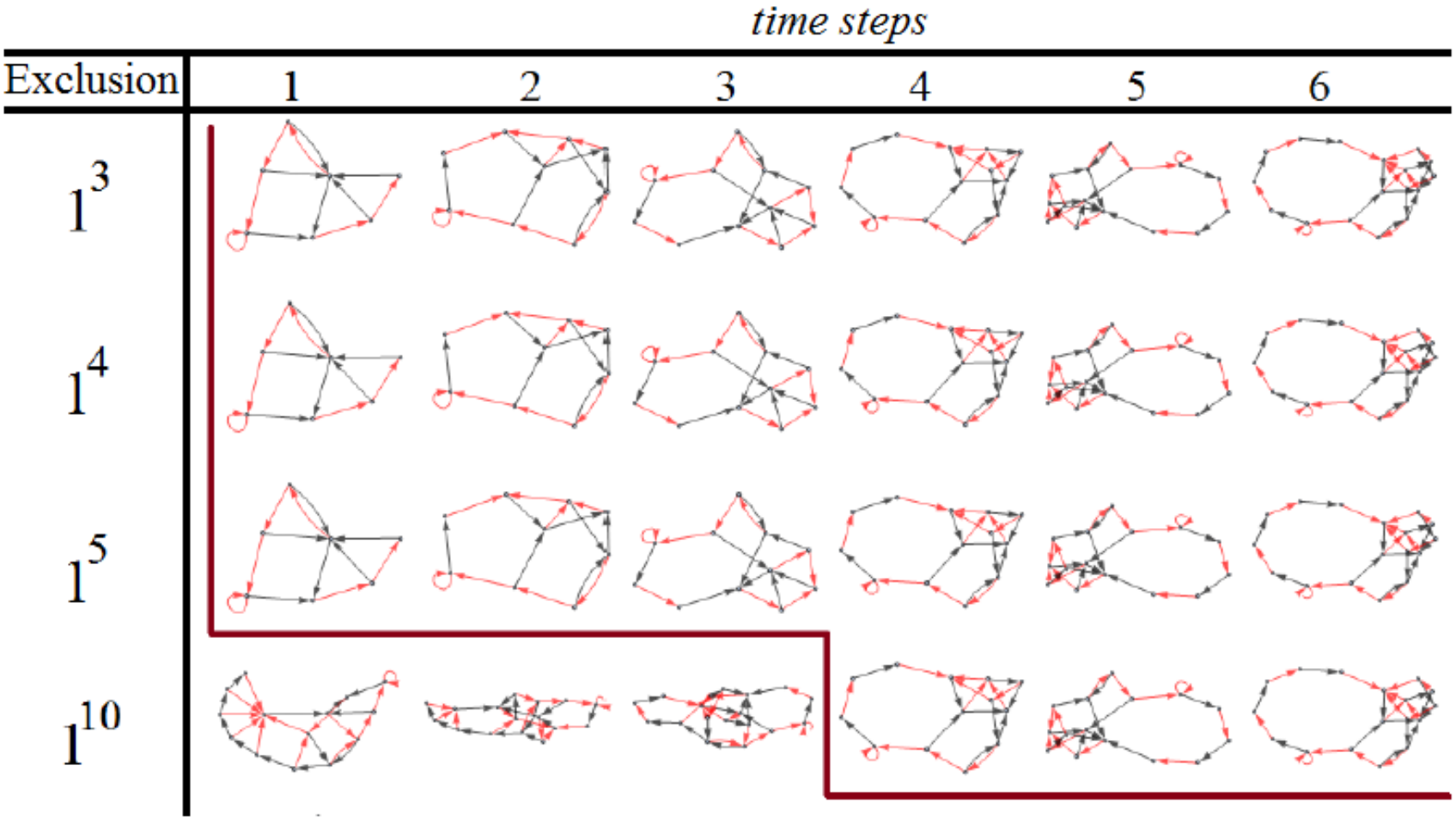
$t = 4$



$t = 5$

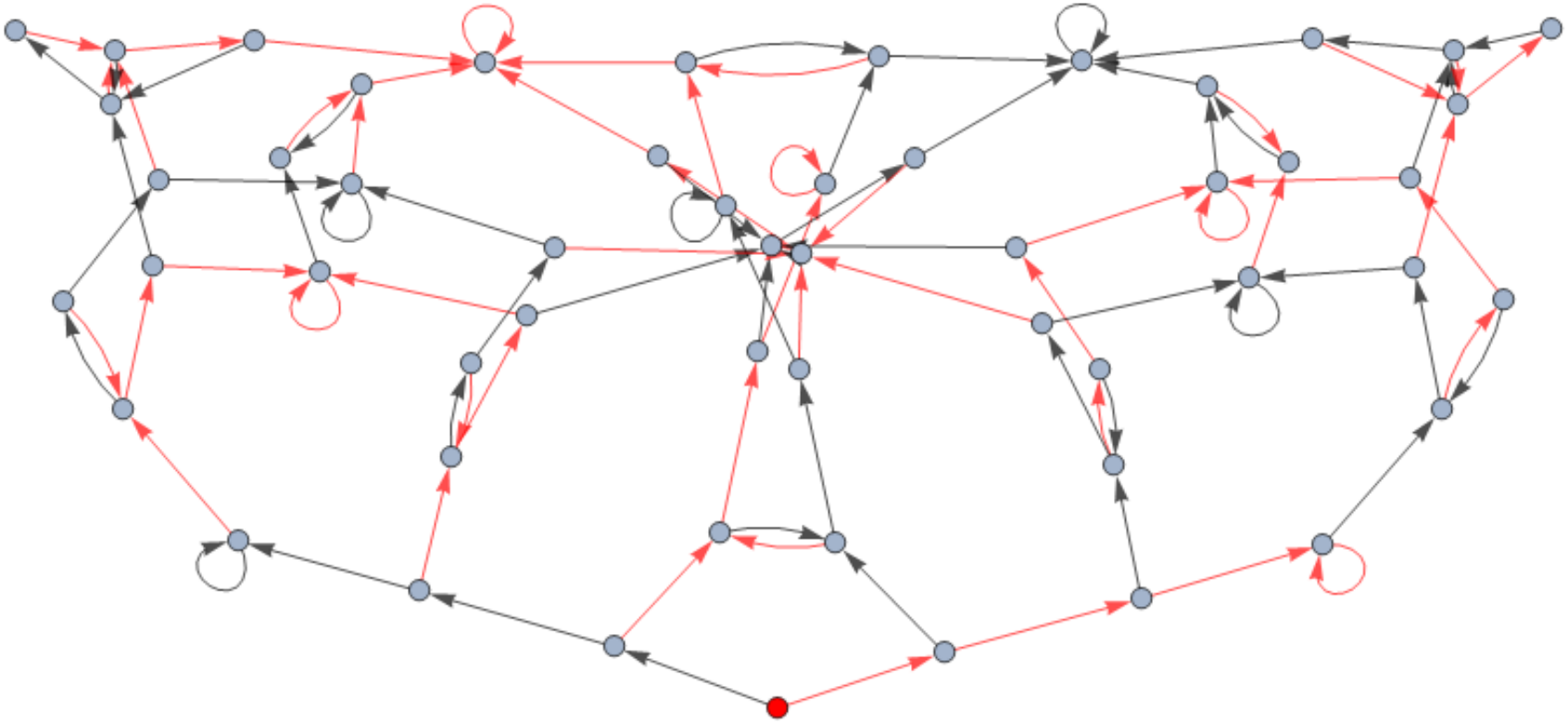
INFERENCE OF LIMIT GRAPHS

Example: Rule 178 – ICs not containing 1^n



INFERENCE OF LIMIT GRAPHS

Example: Rule 178 – limit graph



5. Remarks and next steps

RESULTS

- Wolfram's complexity table was expanded by using the direct method along with alternative functions of minimisation and DFA conversion to process graph. New information on the number of vertices and edges of the graphs for larger time steps for 21 rules were then provided: 11, 14, 18, 22, 23, 26, 37, 41, 43, 50, 54, 56, 94, 122, 134, 140, 142, 146, 162, 164 and 232 (and their dynamical equivalent rules).

- The rules of the ECA rule space were classified according to the growth patterns of their graphs.

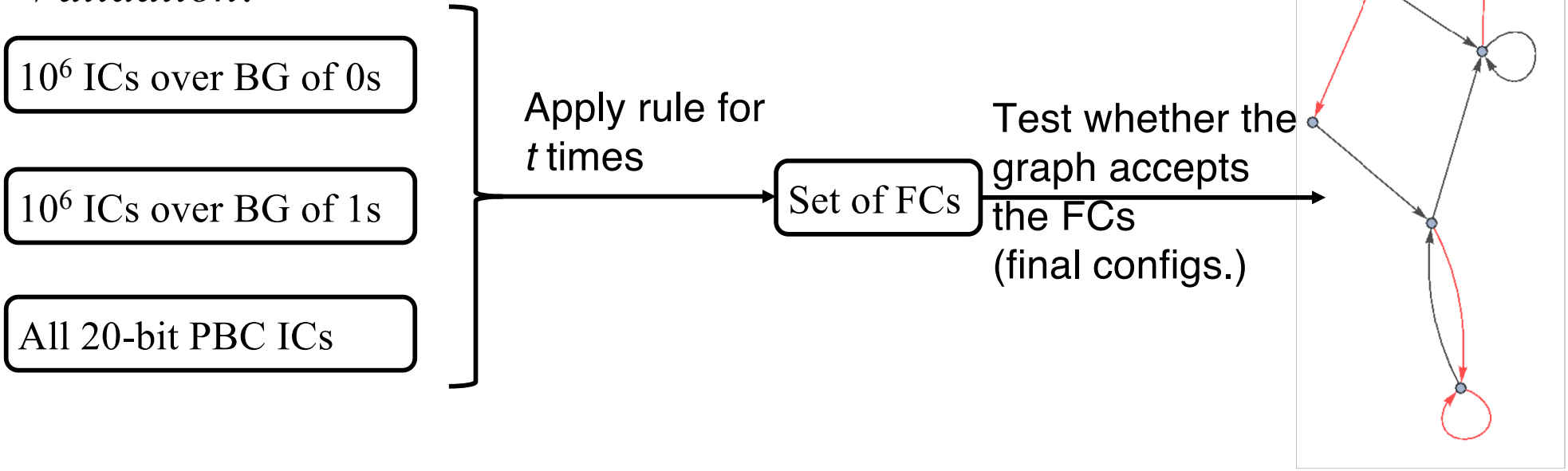
RESULTS

- We were able to infer finite-time graphs for larger values of *t* by using information about their growth patterns for 14 rules (and their dynamical equivalent rules): 23, 32, 40, 56, 128, 132, 136, 140, 142, 162, 168, 172, 184 and 232.

RESULTS

- We were able to infer finite-time graphs for larger values of *t* by using information about their growth patterns for 14 rules (and their dynamical equivalent rules): 23, 32, 40, 56, 128, 132, 136, 140, 142, 162, 168, 172, 184 and 232.

Validation:



RESULTS

- The analysis of the evolution of regular expressions allowed us to infer the limit graphs of rules 32, 56, 128, 132, 136, 140, 168, 178, 184 and 162.

In order to validate the limit graphs, we carried out the same tests applied for the inferred finite-time process graphs.

Also, the limit graph of rule 56 we obtained is the same independently obtained by *Qin & Xie (2005)* by analytical means.

RESULTS

- The limit graphs were used to generate typical limit configurations which in turn allowed us to compute the Fourier spectrum of each rule for nonperiodic configurations. This was previously an issue, since the length of nonperiodic is generally nonconstant and may increase or decrease with time.

E.L.P. Ruivo and P.P.B. de Oliveira. “Computing cellular automata spectra under fixed boundary conditions via limit graphs”. *International Journal of Modern Physics C*, 27(6): 1650073-1/19, 2016.

NEXT STEPS

- Investigate other possible growth patterns in addition to the ones described here, in order to possibly detect more rules with a somehow ordered growth.

- Analyse subsets of initial configurations and how they affect the growth of the process graphs.

- Restrain the set of allowed initial configurations for more complex rules in order to infer subsets of the limit set.