A Faster P Solution for the Byzantine Agreement Problem

Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu
Department of Computer Science, University of Auckland,
Auckland, New Zealand

CMC11, Jena, Germany
23-27 August 2010

Quiz

- What is the most celebrated theoretical result in distributed computing?
- FLP : Fischer, Lynch, and Paterson (1985)
- Impossibility of consensus in asynchronous distributed systems, if there is even one faulty process.
- Proven for both message passing and shared memory asynchronous systems.
- Synchronous systems admit solutions iff $N \geq 3F + 1$.

Quiz

- What is the most celebrated theoretical result in distributed computing?
- FLP : Fischer, Lynch, and Paterson (1985)
- Impossibility of consensus in asynchronous distributed systems, if there is even one faulty process.
- Proven for both message passing and shared memory asynchronous systems.
- Synchronous systems admit solutions iff $N \geq 3F + 1$.

Quiz

- What is the most celebrated theoretical result in distributed computing?
- FLP : Fischer, Lynch, and Paterson (1985)
- Impossibility of consensus in asynchronous distributed systems, if there is even one faulty process.
- Proven for both message passing and shared memory asynchronous systems.
- Synchronous systems admit solutions iff $N \geq 3F + 1$.

Quiz

- What is the most celebrated theoretical result in distributed computing?
- FLP : Fischer, Lynch, and Paterson (1985)
- Impossibility of consensus in asynchronous distributed systems, if there is even one faulty process.
- Proven for both message passing and shared memory asynchronous systems.
- Synchronous systems admit solutions iff $N \geq 3F + 1$.

Quiz

- What is the most celebrated theoretical result in distributed computing?
- FLP : Fischer, Lynch, and Paterson (1985)
- Impossibility of consensus in asynchronous distributed systems, if there is even one faulty process.
- Proven for both message passing and shared memory asynchronous systems.
- Synchronous systems admit solutions iff $N \geq 3F + 1$.

Quiz

- What is the most celebrated theoretical result in distributed computing?
- FLP : Fischer, Lynch, and Paterson (1985)
- Impossibility of consensus in asynchronous distributed systems, if there is even one faulty process.
- Proven for both message passing and shared memory asynchronous systems.
- Synchronous systems admit solutions iff $N \geq 3F + 1$.

Motivation

- P systems are a highly parallel and distributed computing model.
- Can we apply P systems to solve complex problems from distributed computing, such as the Byzantine agreement?
- Will the the P system solution compare favorably with the classical solution: performance, resources, expressiveness.
- Can we provide feedback on P systems programmability (features required or beneficial in modeling complex systems)?
- Can we formulate a native P systems solution?

Motivation

- P systems are a highly parallel and distributed computing model.
- Can we apply P systems to solve complex problems from distributed computing, such as the Byzantine agreement?
- Will the the P system solution compare favorably with the classical solution: performance, resources, expressiveness.
- Can we provide feedback on P systems programmability (features required or beneficial in modeling complex systems)?
- Can we formulate a native P systems solution?

Motivation

- P systems are a highly parallel and distributed computing model.
- Can we apply P systems to solve complex problems from distributed computing, such as the Byzantine agreement?
- Will the the P system solution compare favorably with the classical solution: performance, resources, expressiveness.
- Can we provide feedback on P systems programmability (features required or beneficial in modeling complex systems)?
- Can we formulate a native P systems solution?

Motivation

- P systems are a highly parallel and distributed computing model.
- Can we apply P systems to solve complex problems from distributed computing, such as the Byzantine agreement?
- Will the the P system solution compare favorably with the classical solution: performance, resources, expressiveness.
- Can we provide feedback on P systems programmability (features required or beneficial in modeling complex systems)?
- Can we formulate a native P systems solution?

Motivation

- P systems are a highly parallel and distributed computing model.
- Can we apply P systems to solve complex problems from distributed computing, such as the Byzantine agreement?
- Will the the P system solution compare favorably with the classical solution: performance, resources, expressiveness.
- Can we provide feedback on P systems programmability (features required or beneficial in modeling complex systems)?
- Can we formulate a native P systems solution?

Our work—Bird's eye view

- We have earlier [JLAP, 2010] proposed a first P solution, based on EIG trees.
- Here [CMC11, 2010], we propose an improved solution, which uses less cells and runs faster.
- The following table compares [CMC11, 2010] with [JLAP, 2010] (typically, $L = \lceil N/3 \rceil$).

Criterion	JLAP-2010	CMC11-2010
# P steps	$9L + 6$	$6L + 1$
# cells per process	$2N + 1 + O(N!)$	$3N + 1$
type of channels	duplex and simplex	duplex

- For comparison, the standard EIG solution runs in L messaging steps plus 1 more big evaluation step.

Our work—Bird's eye view

- We have earlier [JLAP, 2010] proposed a first P solution, based on EIG trees.
- Here [CMC11, 2010], we propose an improved solution, which uses less cells and runs faster.
- The following table compares [CMC11, 2010] with [JLAP, 2010] (typically, $L = \lceil N/3 \rceil$).

Criterion	JLAP-2010	CMC11-2010
# P steps	$9L + 6$	$6L + 1$
# cells per process	$2N + 1 + O(N!)$	$3N + 1$
type of channels	duplex and simplex	duplex

- For comparison, the standard EIG solution runs in L messaging steps plus 1 more big evaluation step.

Our work—Bird's eye view

- We have earlier [JLAP, 2010] proposed a first P solution, based on EIG trees.
- Here [CMC11, 2010], we propose an improved solution, which uses less cells and runs faster.
- The following table compares [CMC11, 2010] with [JLAP, 2010] (typically, $L = \lceil N/3 \rceil$).

Criterion	JLAP-2010	CMC11-2010
# P steps	$9L + 6$	$6L + 1$
# cells per process	$2N + 1 + O(N!)$	$3N + 1$
type of channels	duplex and simplex	duplex

- For comparison, the standard EIG solution runs in L messaging steps plus 1 more big evaluation step.

Our work—Bird's eye view

- We have earlier [JLAP, 2010] proposed a first P solution, based on EIG trees.
- Here [CMC11, 2010], we propose an improved solution, which uses less cells and runs faster.
- The following table compares [CMC11, 2010] with [JLAP, 2010] (typically, $L = \lceil N/3 \rceil$).

Criterion	**JLAP-2010**	**CMC11-2010**
# P steps	$9L + 6$	$6L + 1$
# cells per process	$2N + 1 + O(N!)$	$3N + 1$
type of channels	duplex and simplex	duplex

- For comparison, the standard EIG solution runs in L messaging steps plus 1 more big evaluation step.

Our work—Bird's eye view

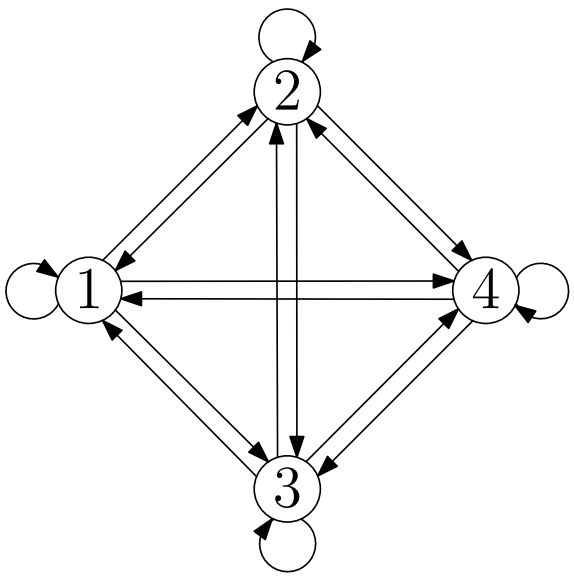
- We have earlier [JLAP, 2010] proposed a first P solution, based on EIG trees.
- Here [CMC11, 2010], we propose an improved solution, which uses less cells and runs faster.
- The following table compares [CMC11, 2010] with [JLAP, 2010] (typically, $L = \lceil N/3 \rceil$).

Criterion	**JLAP-2010**	**CMC11-2010**
# P steps	$9L + 6$	$6L + 1$
# cells per process	$2N + 1 + O(N!)$	$3N + 1$
type of channels	duplex and simplex	duplex

- For comparison, the standard EIG solution runs in L messaging steps plus 1 more big evaluation step.

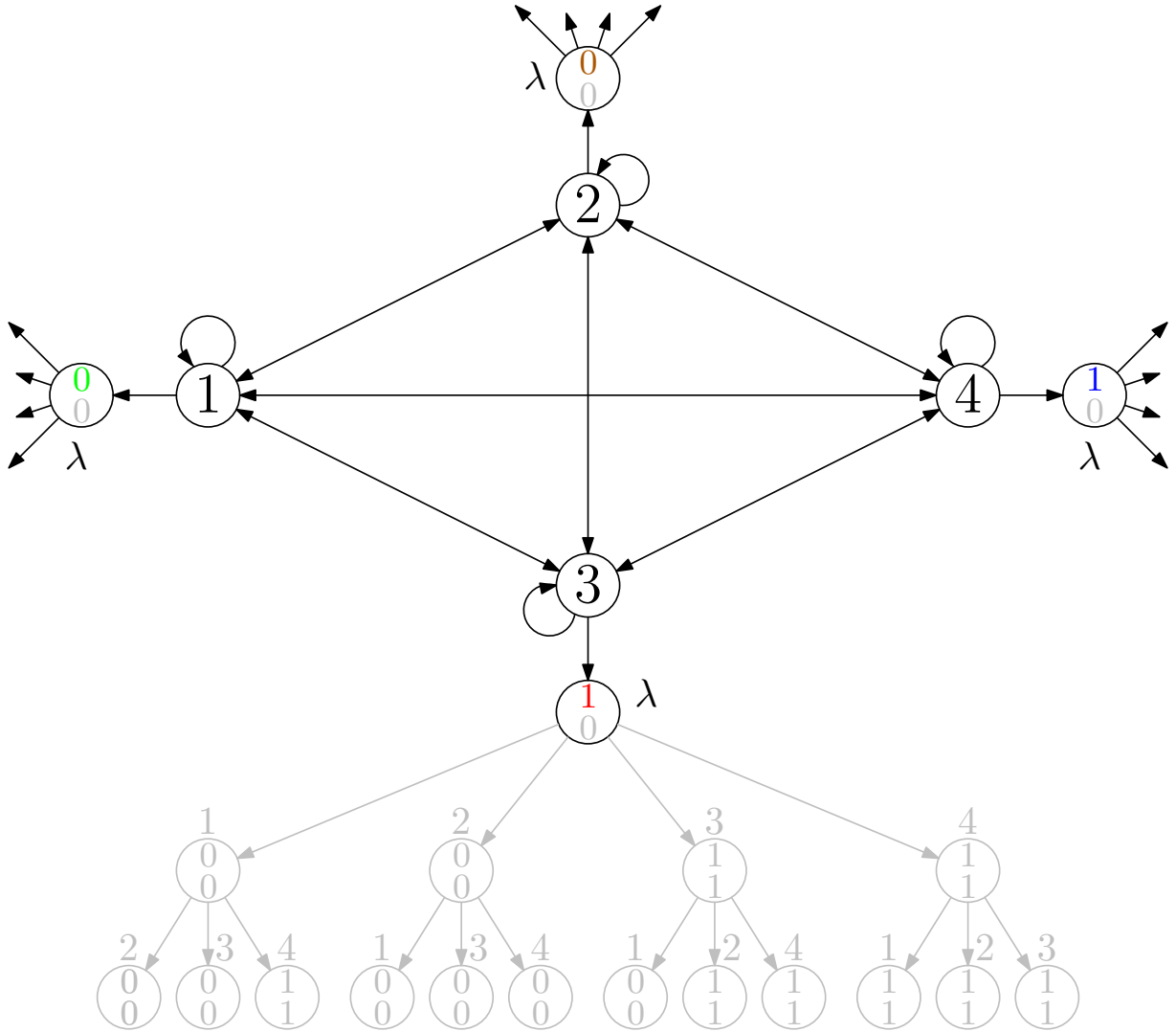
A sample Byzantine scenario, $N = 4$

Complete graph, synchronous model.

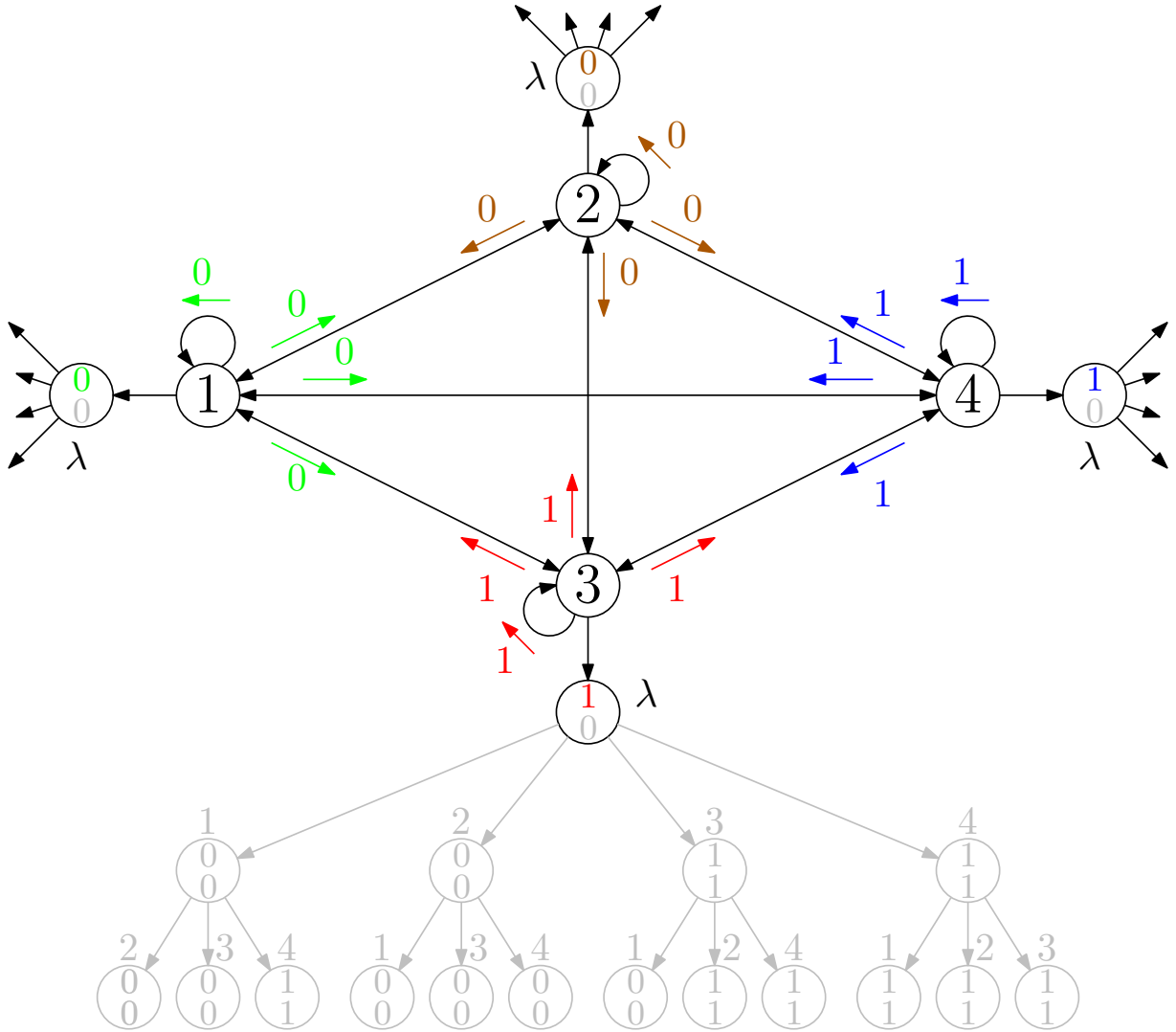


Process	1	2	3	4
Initial choice	0	0	1	1
Faulty	Yes	No	No	No
Round 1 messages	(λ, x)	$(\lambda, 0)$	$(\lambda, 1)$	$(\lambda, 1)$
Round 2 messages	$(1,0)(2,0)$ $(3,y)(4,1)$	$(1,0)(2,0)$ $(3,1)(4,1)$	$(1,0)(2,0)$ $(3,1)(4,1)$	$(1,1)(2,0)$ $(3,1)(4,1)$
Final decision	?	0	0	0

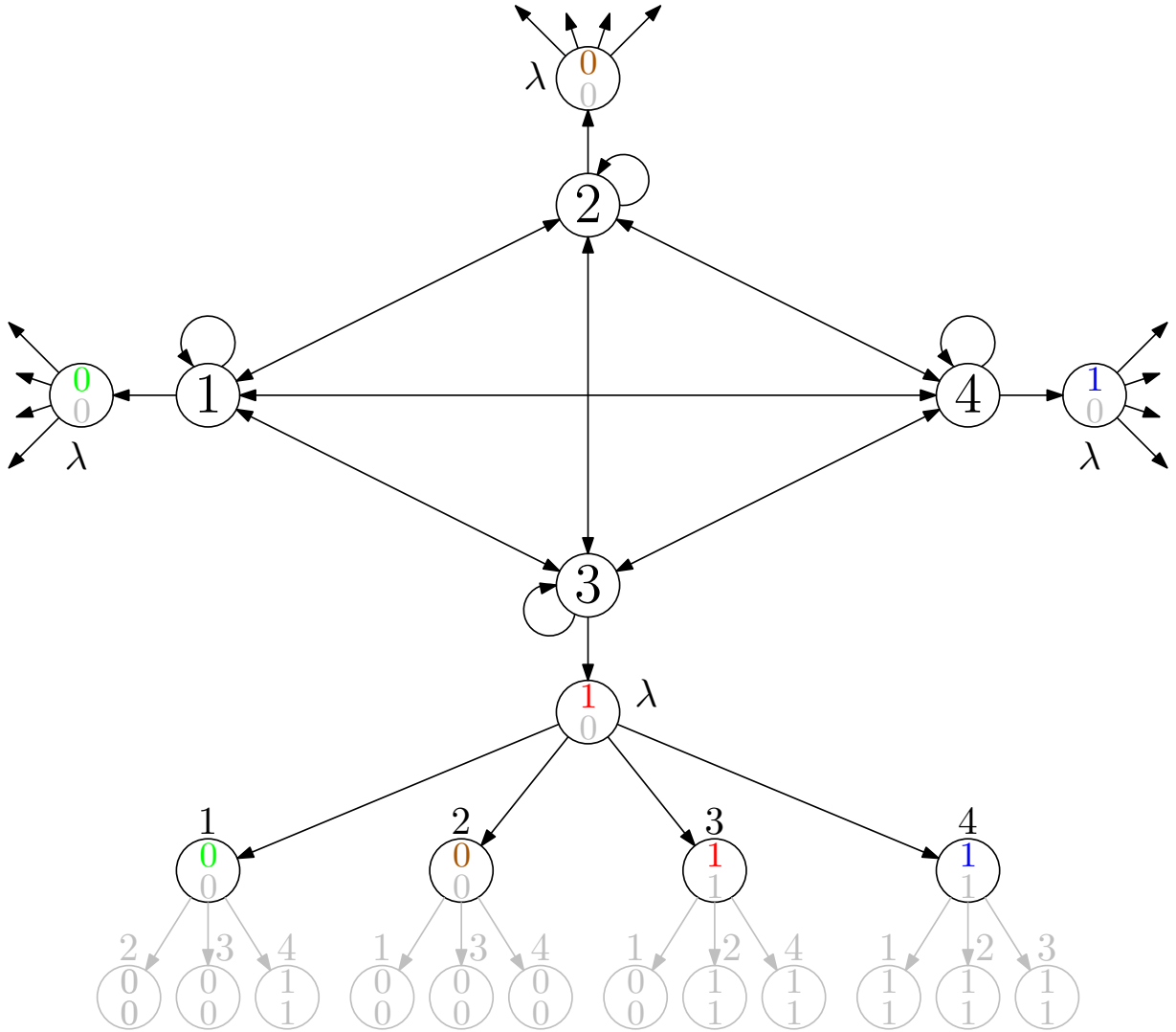
EIG messaging phase



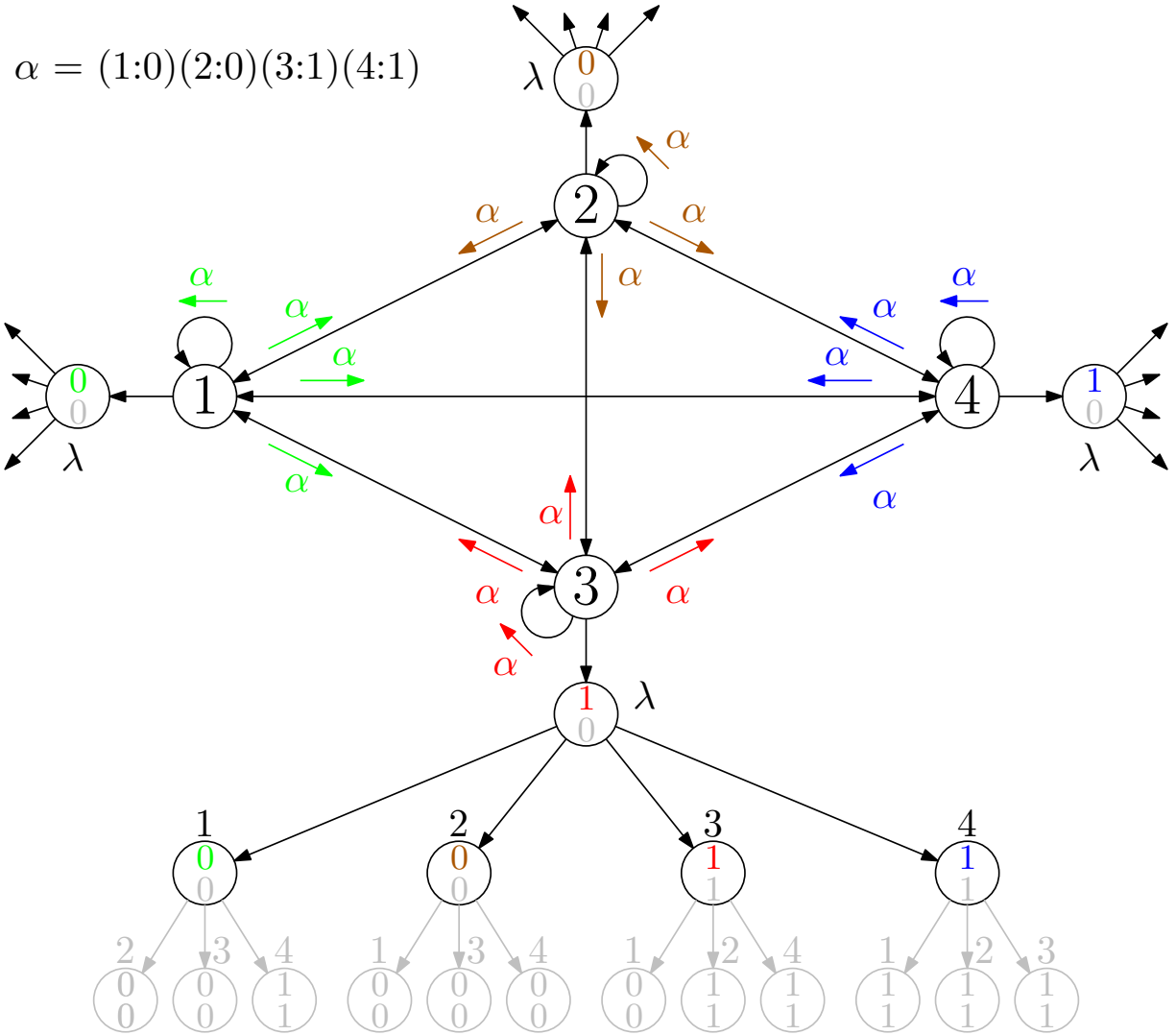
EIG messaging phase



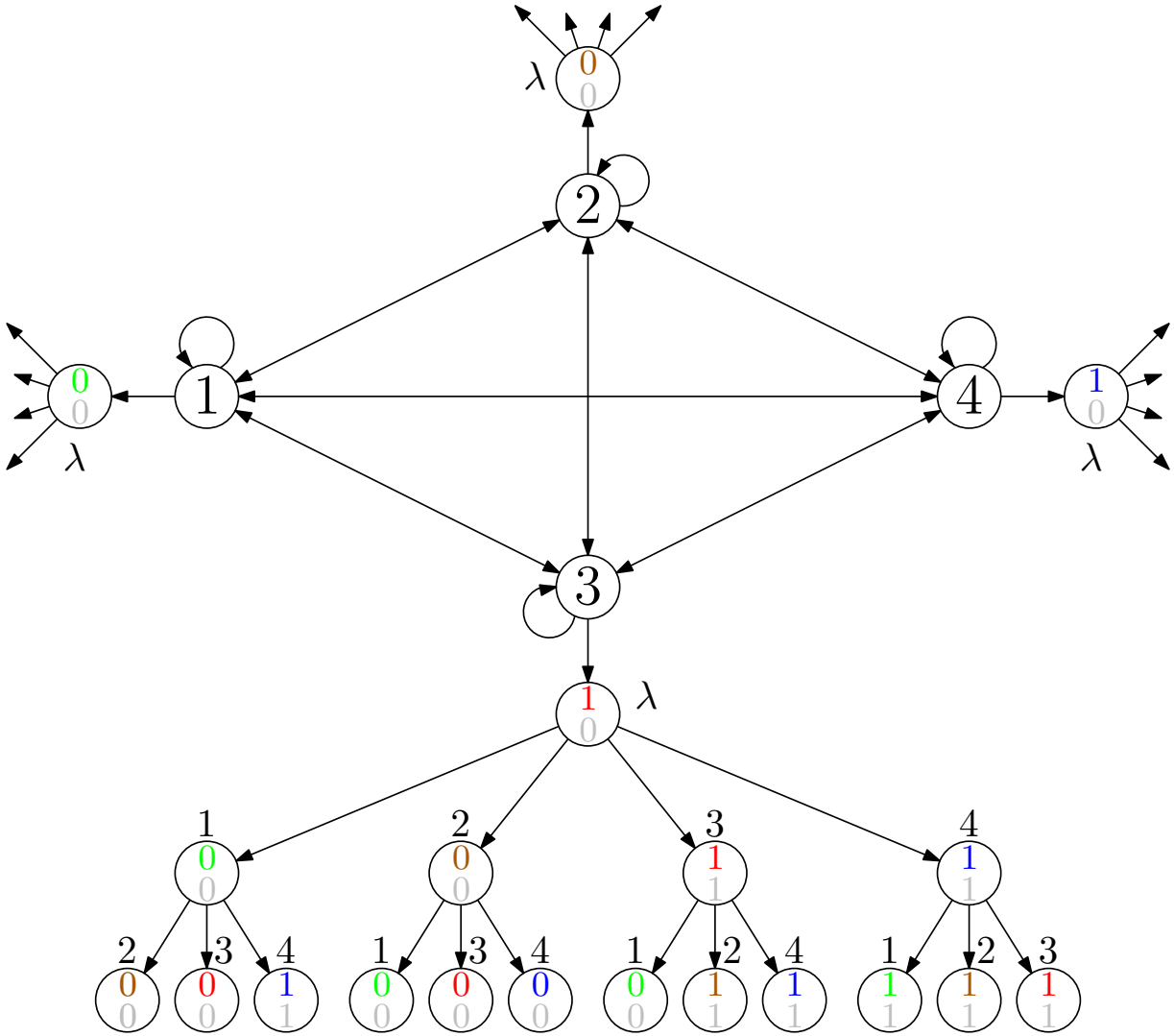
EIG messaging phase



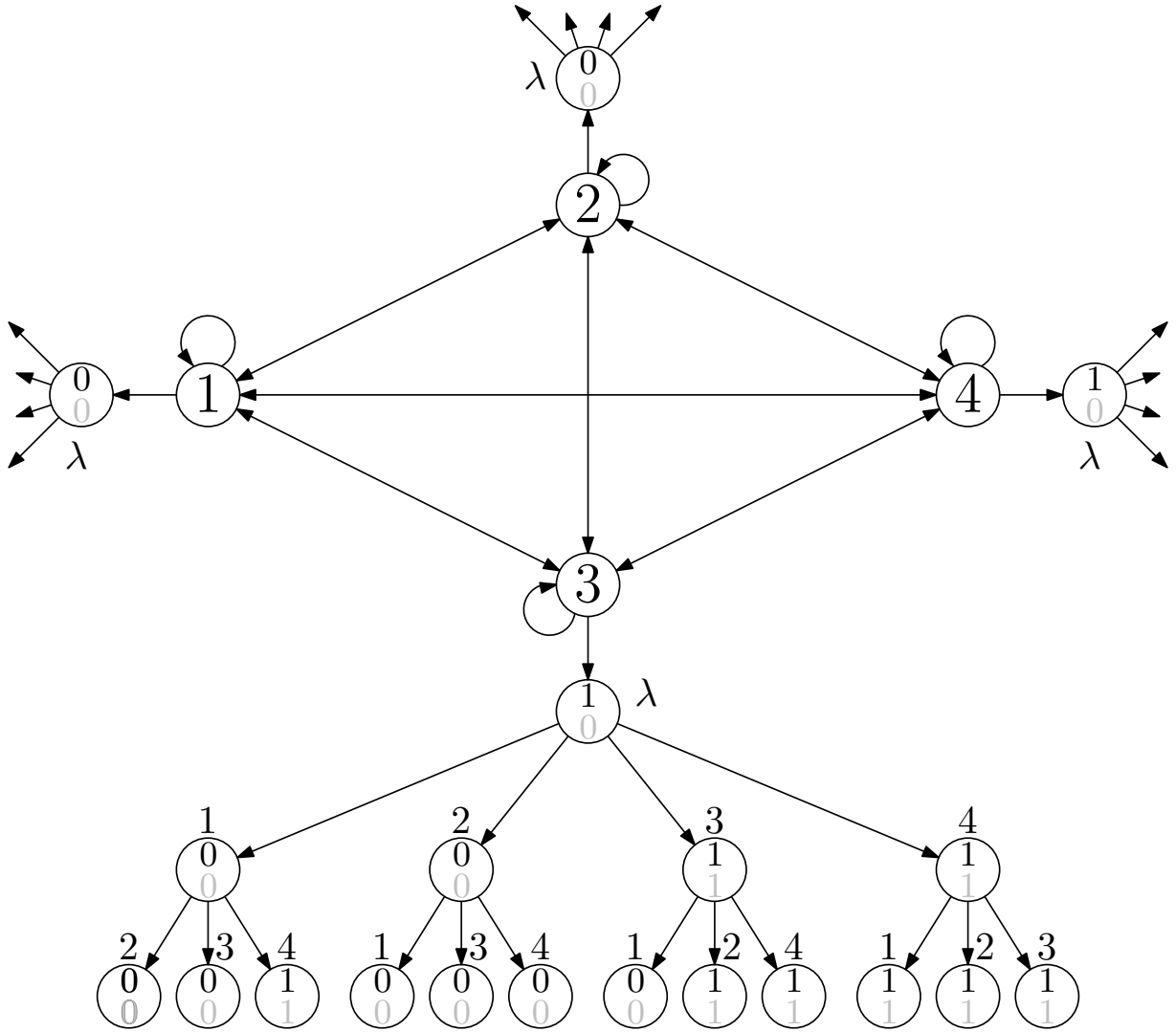
EIG messaging phase



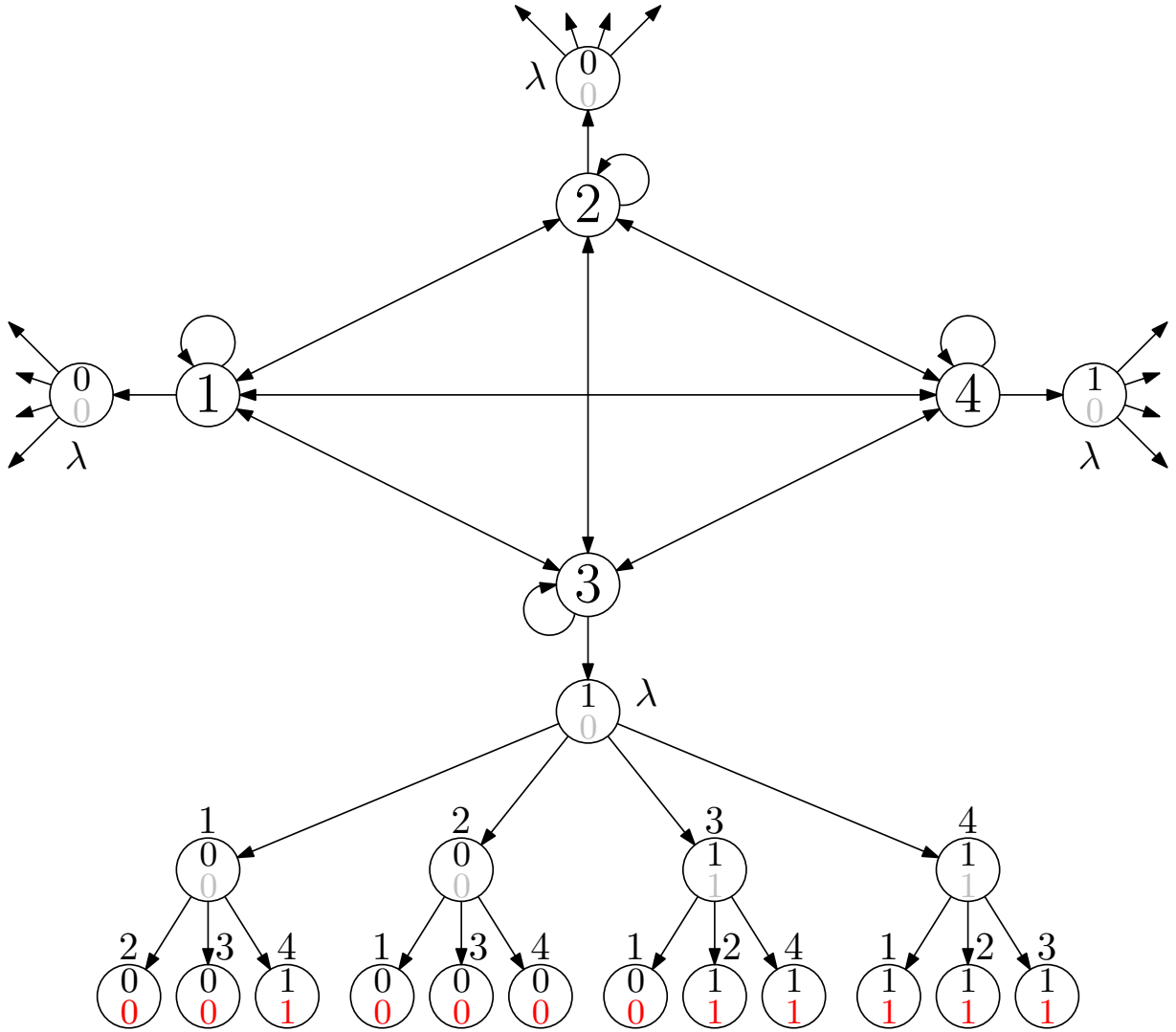
EIG messaging phase



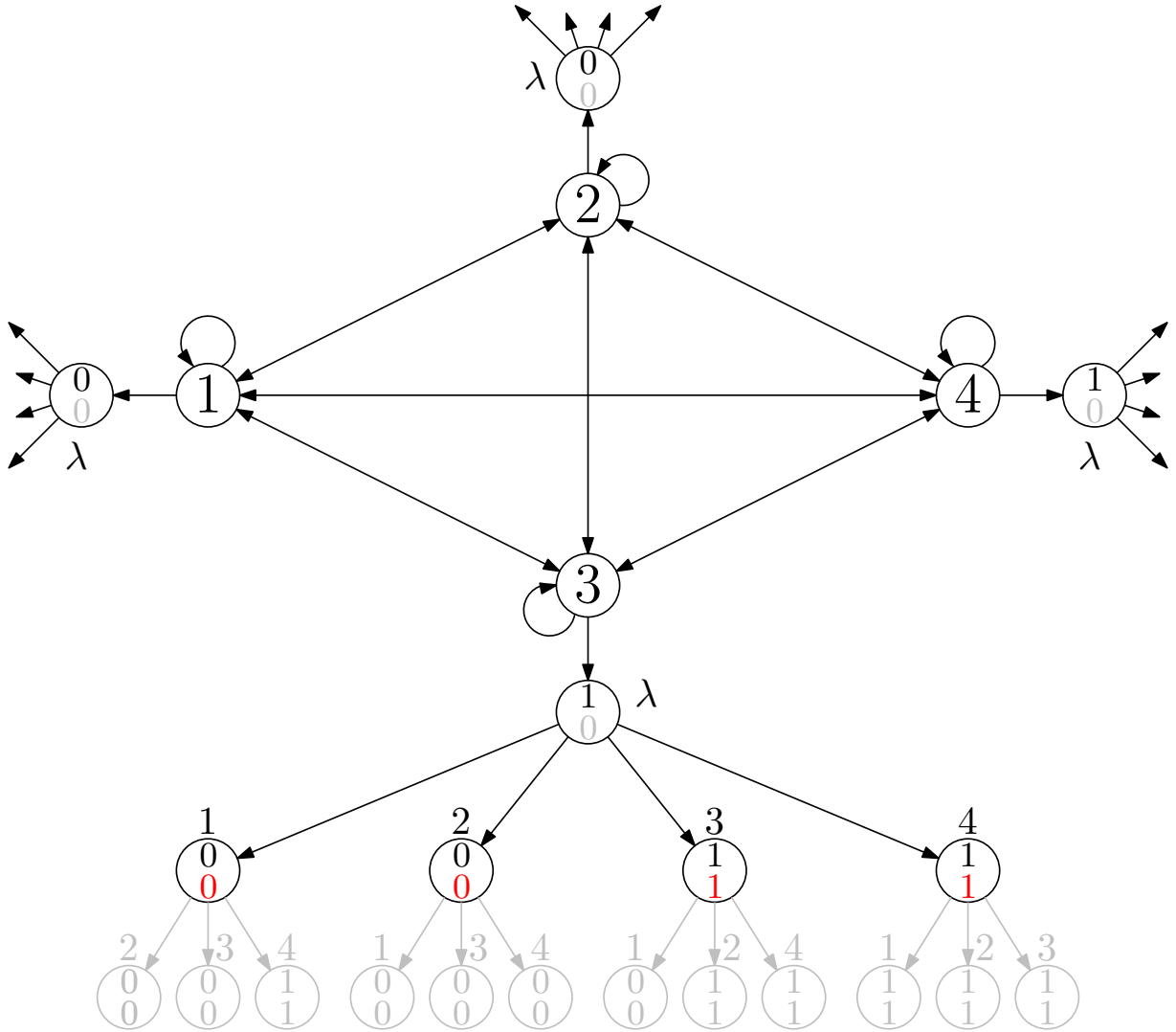
EIG evaluation phase



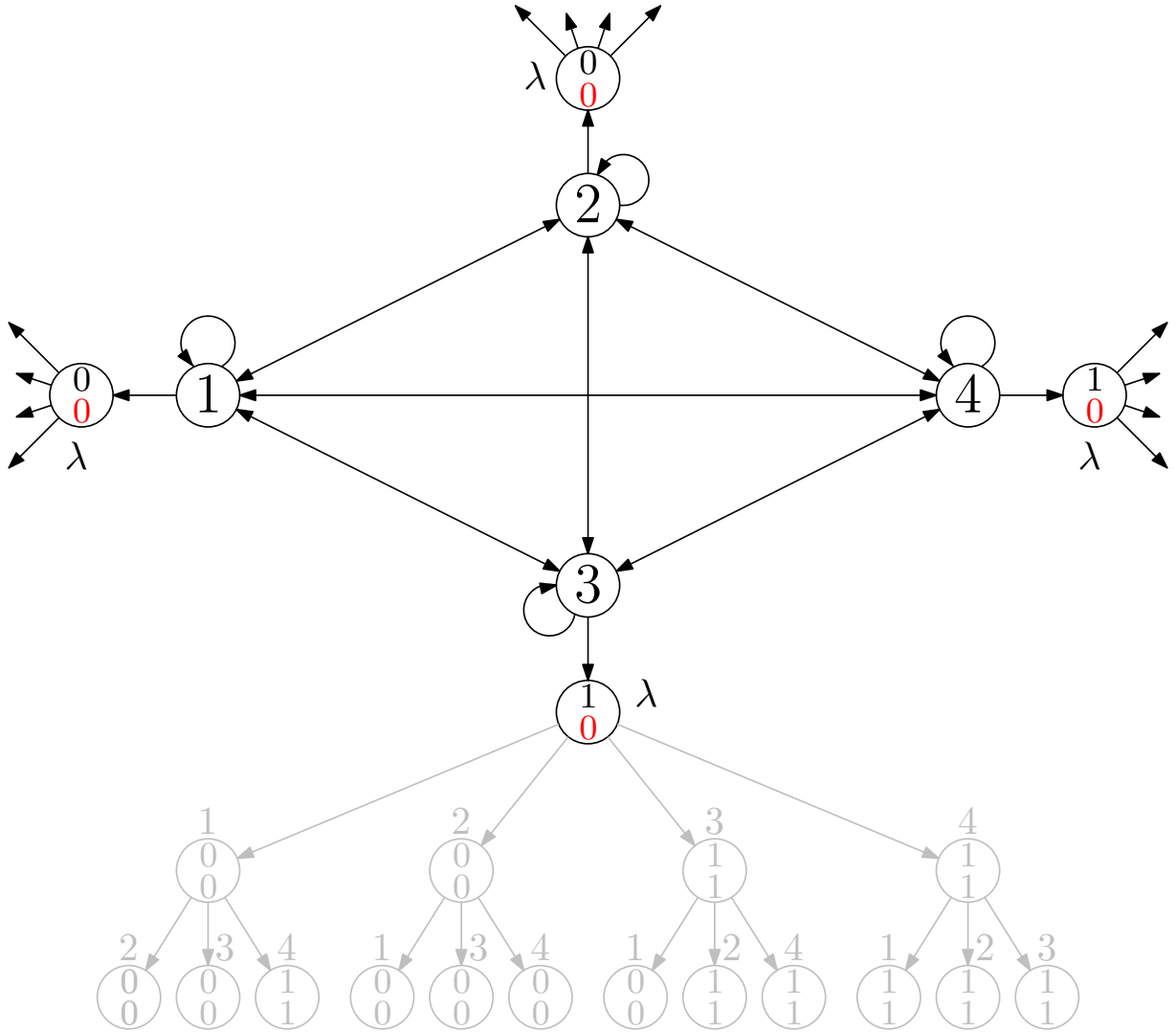
EIG evaluation phase



EIG evaluation phase



EIG evaluation phase



P modules

- To support our complex design, in [JLAP-2010] we have proposed a new modular framework, called P modules, that supports generic objects, encapsulation, information hiding and recursive composition.
- In [CMC11-2010], we extend our previous proposal, with external definitions and external references, which support cleaner and safer module interconnection facilities.

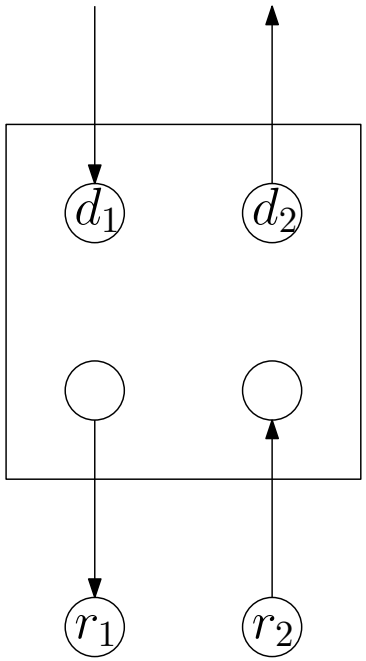
P modules

- To support our complex design, in [JLAP-2010] we have proposed a new modular framework, called P modules, that supports generic objects, encapsulation, information hiding and recursive composition.
- In [CMC11-2010], we extend our previous proposal, with external definitions and external references, which support cleaner and safer module interconnection facilities.

P modules

- To support our complex design, in [JLAP-2010] we have proposed a new modular framework, called P modules, that supports generic objects, encapsulation, information hiding and recursive composition.
- In [CMC11-2010], we extend our previous proposal, with external definitions and external references, which support cleaner and safer module interconnection facilities.

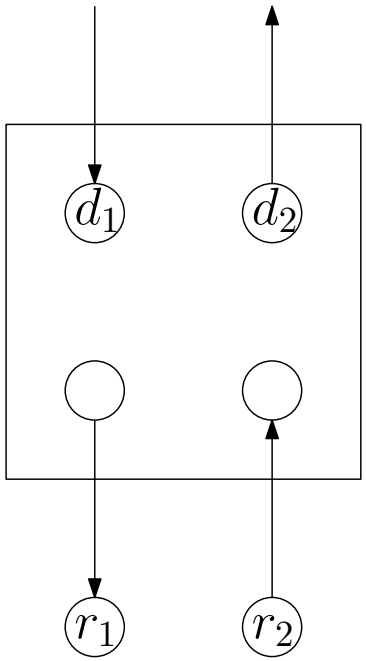
P Module—Generic parameters: ref, def, sym



$d_1 \in D_\downarrow$ (external $def_\downarrow$)
$d_2 \in D_\uparrow$ (external $def_\uparrow$)
$r_1 \in R_\downarrow$ (external $ref_\downarrow$)
$r_2 \in R_\uparrow$ (external $ref_\uparrow$)

- $\texttt{ref}_\downarrow$ and $\texttt{ref}_\uparrow$ are instantiated when this module connects to another module
- $\texttt{def}_\downarrow$ and $\texttt{def}_\uparrow$ are instantiated when another module connects to this module
- (not shown here) `sym` are instantiated to actual objects, as needed

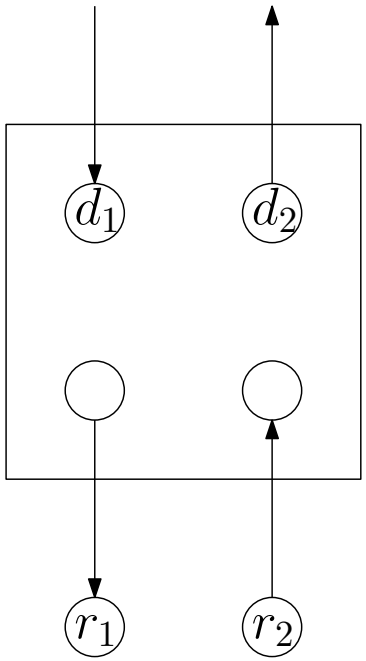
P Module—Generic parameters: ref, def, sym



$d_1 \in D_\downarrow$ (external $def_\downarrow$)
$d_2 \in D_\uparrow$ (external $def_\uparrow$)
$r_1 \in R_\downarrow$ (external $ref_\downarrow$)
$r_2 \in R_\uparrow$ (external $ref_\uparrow$)

- $\texttt{ref}_\downarrow$ and $\texttt{ref}_\uparrow$ are instantiated when this module connects to another module
- $\texttt{def}_\downarrow$ and $\texttt{def}_\uparrow$ are instantiated when another module connects to this module
- (not shown here) `sym` are instantiated to actual objects, as needed

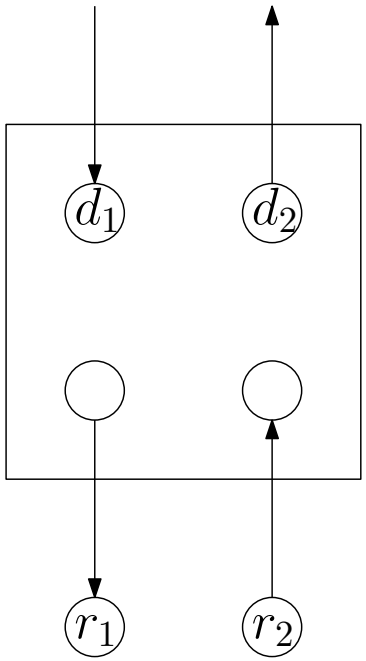
P Module—Generic parameters: ref, def, sym



$d_1 \in D_\downarrow$ (external $def_\downarrow$)
$d_2 \in D_\uparrow$ (external $def_\uparrow$)
$r_1 \in R_\downarrow$ (external $ref_\downarrow$)
$r_2 \in R_\uparrow$ (external $ref_\uparrow$)

- $\texttt{ref}_\downarrow$ and $\texttt{ref}_\uparrow$ are instantiated when this module connects to another module
- $\texttt{def}_\downarrow$ and $\texttt{def}_\uparrow$ are instantiated when another module connects to this module
- (not shown here) sym are instantiated to actual objects, as needed

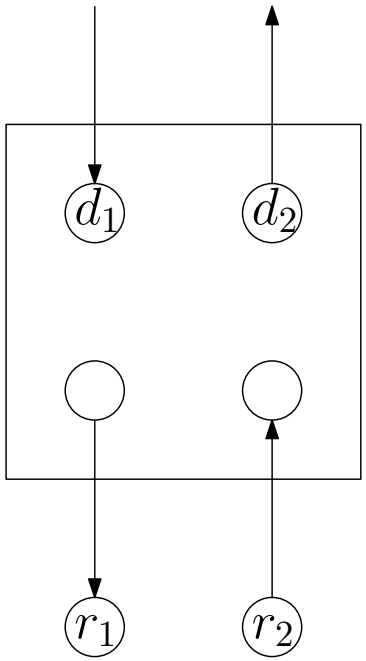
P Module—Generic parameters: ref, def, sym



$d_1 \in D_\downarrow$ (external $def_\downarrow$)
$d_2 \in D_\uparrow$ (external $def_\uparrow$)
$r_1 \in R_\downarrow$ (external $ref_\downarrow$)
$r_2 \in R_\uparrow$ (external $ref_\uparrow$)

- $\texttt{ref}_\downarrow$ and $\texttt{ref}_\uparrow$ are instantiated when this module connects to another module
- $\texttt{def}_\downarrow$ and $\texttt{def}_\uparrow$ are instantiated when another module connects to this module
- (not shown here) `sym` are instantiated to actual objects, as needed

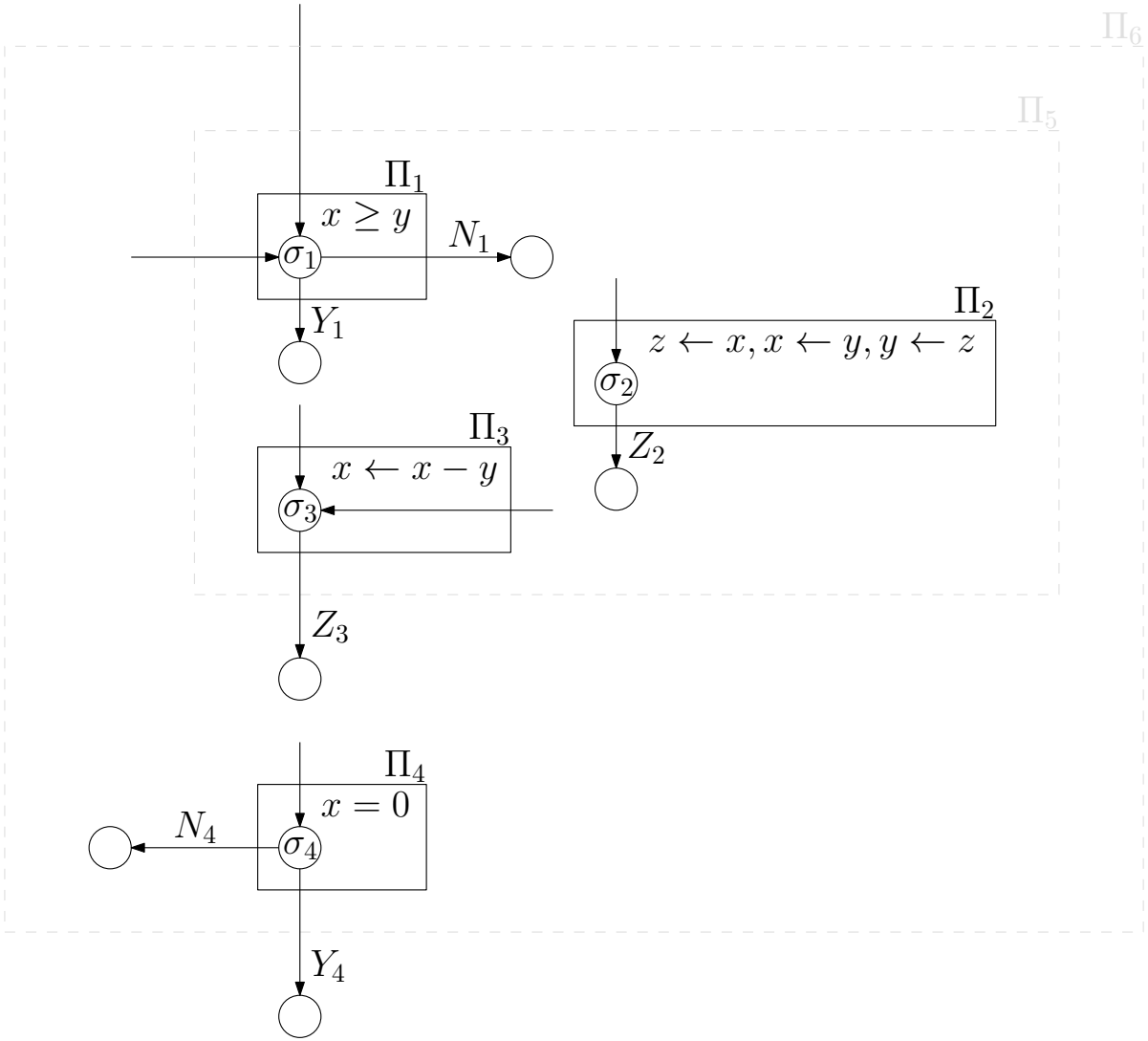
P Module—Generic parameters: ref, def, sym



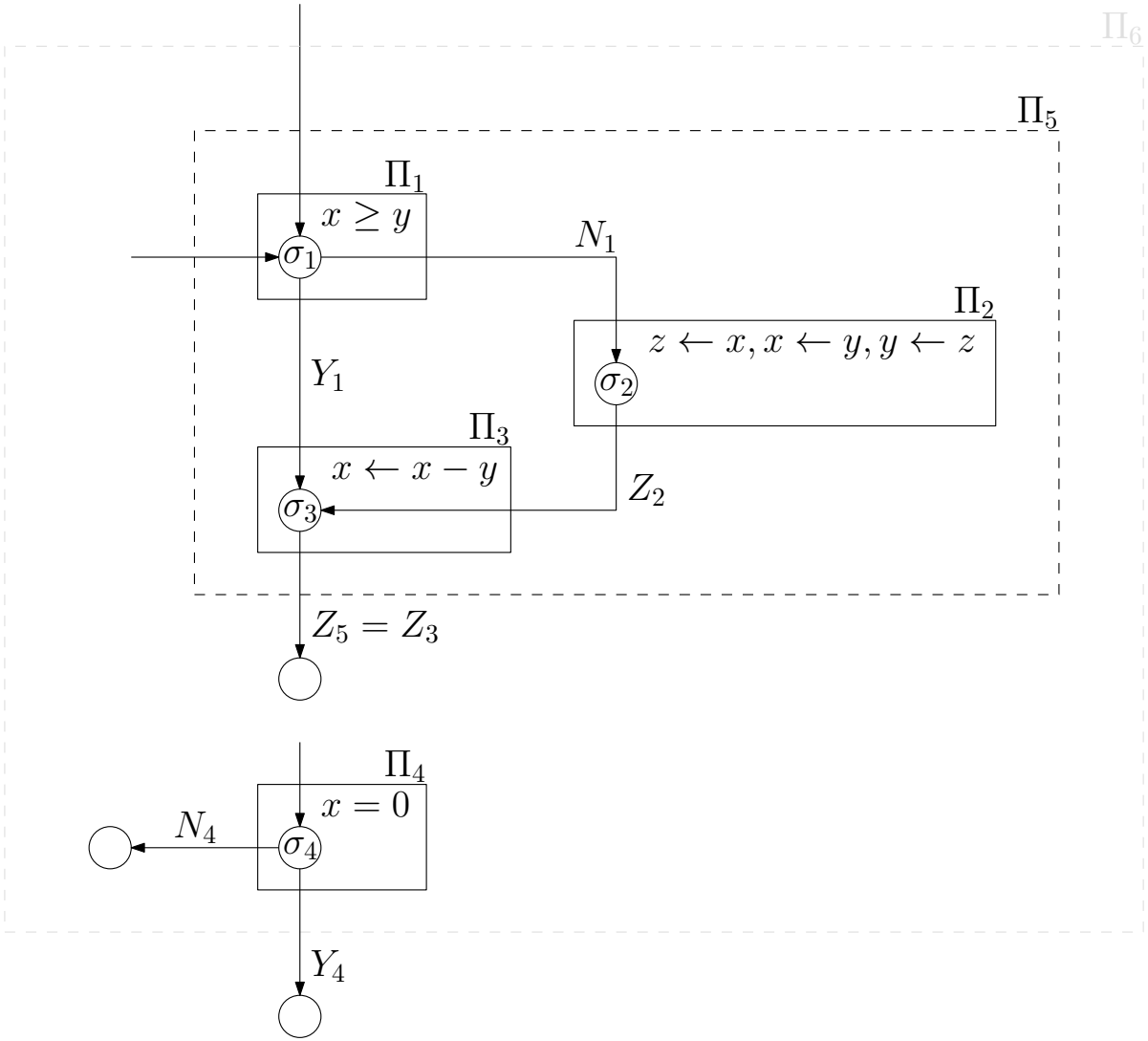
$d_1 \in D_\downarrow$ (external $def_\downarrow$)
$d_2 \in D_\uparrow$ (external $def_\uparrow$)
$r_1 \in R_\downarrow$ (external $ref_\downarrow$)
$r_2 \in R_\uparrow$ (external $ref_\uparrow$)

- $\texttt{ref}_\downarrow$ and $\texttt{ref}_\uparrow$ are instantiated when this module connects to another module
- $\texttt{def}_\downarrow$ and $\texttt{def}_\uparrow$ are instantiated when another module connects to this module
- (not shown here) `sym` are instantiated to actual objects, as needed

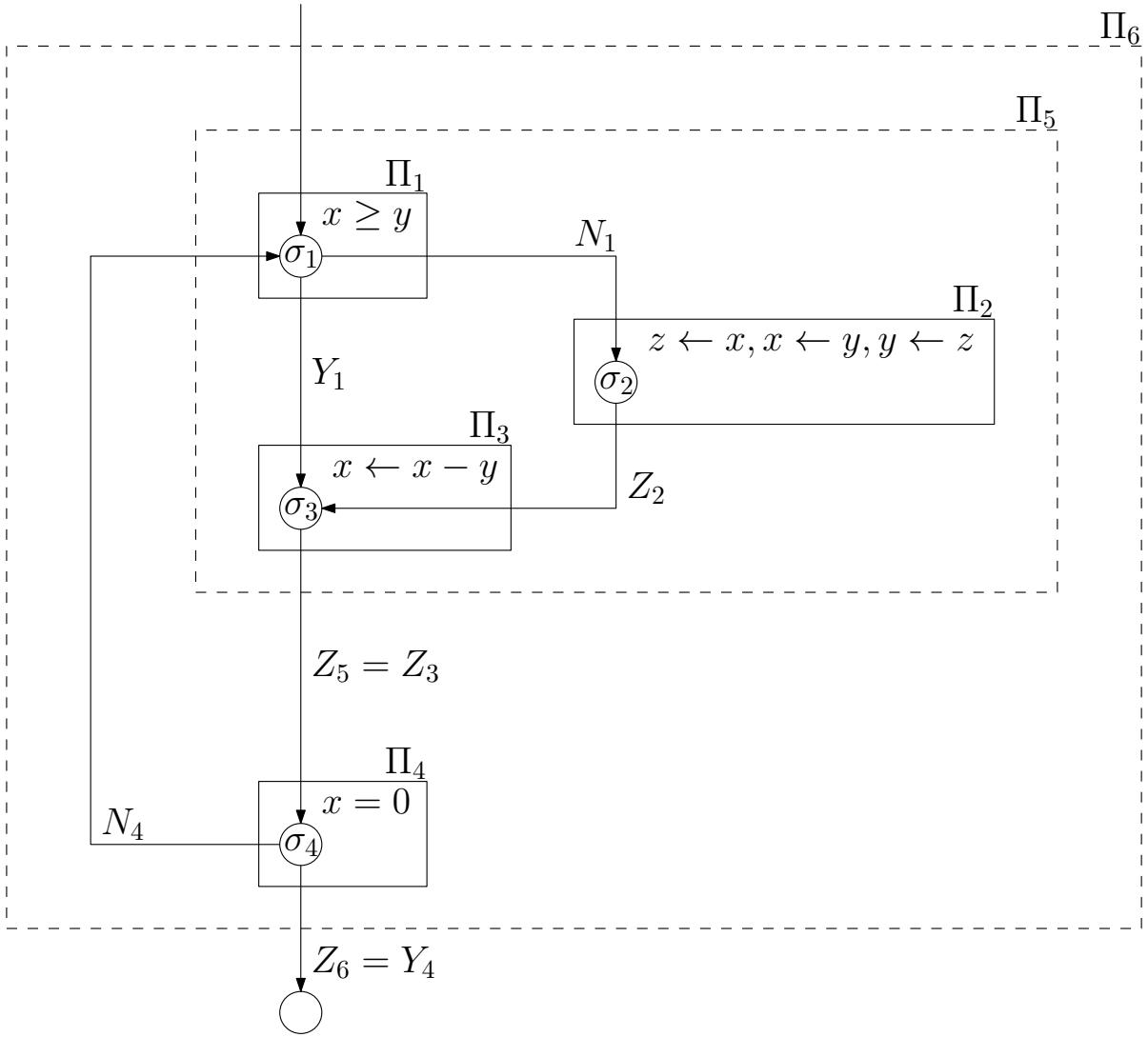
Example—Modular composition of a GCD system



Example—Modular composition of a GCD system



Example—Modular composition of a GCD system



P module definition

Definition (P module)

A P module is a system $\Pi = (O, K, \delta, S, D_{\uparrow}, D_{\downarrow}, R_{\uparrow}, R_{\downarrow})$, where:

- O is a finite non-empty alphabet of objects;
- K is a finite set of cells;
- δ is a subset of $(K \times K) \cup (K \times R_{\downarrow}) \cup (R_{\uparrow} \times K)$, i.e. a set of parent-child structural arcs, representing duplex or simplex communication channels, between two existing cells or between an existing cell and an external reference;
- S is a finite alphabet, disjoint of O, of generic `sym` objects;
- $D_{\uparrow}$ is a subset of K, representing $\texttt{def}_{\uparrow}$ definitions;
- $D_{\downarrow}$ is a subset of K, representing $\texttt{def}_{\downarrow}$ definitions;
- $R_{\uparrow}$ is a finite set, disjoint of K, representing $\texttt{ref}_{\uparrow}$ references;
- $R_{\downarrow}$ is a finite set, disjoint of K, representing $\texttt{ref}_{\downarrow}$ references.

...

P module rules

- Rules are applied in a weak-priority order, ensuring a kind of preemption (which we believe essential for complex systems).
- Typical cases, for a cell σ, in state s and containing aa:
  - $s\ a \rightarrow_{\texttt{min}} s'\ b\ (c)_{\uparrow_{\texttt{repl}}}$ can be applied once
  - $s\ a \rightarrow_{\texttt{max}} s'\ b\ (c)_{\uparrow_{\texttt{repl}}}$ can be applied twice
  - $s\ a \rightarrow_{\texttt{min}} s'\ b\ (c)_{\downarrow_{\sigma'}}$ (where $\sigma' \in K$) can be applied once
  - $s\ a \rightarrow_{\texttt{max}} s'\ b\ (c)_{\downarrow_{\sigma'}}$ (where $\sigma' \in K$) can be applied twice

P module rules

- Rules are applied in a weak-priority order, ensuring a kind of preemption (which we believe essential for complex systems).
- Typical cases, for a cell σ, in state s and containing aa:
  - $s\ a \rightarrow_{\texttt{min}} s'\ b\ (c)_{\uparrow_{\texttt{repl}}}$ can be applied once
  - $s\ a \rightarrow_{\texttt{max}} s'\ b\ (c)_{\uparrow_{\texttt{repl}}}$ can be applied twice
  - $s\ a \rightarrow_{\texttt{min}} s'\ b\ (c)_{\downarrow_{\sigma'}}$ (where $\sigma' \in K$) can be applied once
  - $s\ a \rightarrow_{\texttt{max}} s'\ b\ (c)_{\downarrow_{\sigma'}}$ (where $\sigma' \in K$) can be applied twice

P module rules

- Rules are applied in a weak-priority order, ensuring a kind of preemption (which we believe essential for complex systems).
- Typical cases, for a cell σ, in state s and containing aa:
  - $s\ a \rightarrow_{\texttt{min}} s'\ b\ (c)_{\uparrow_{\texttt{repl}}}$ can be applied once
  - $s\ a \rightarrow_{\texttt{max}} s'\ b\ (c)_{\uparrow_{\texttt{repl}}}$ can be applied twice
  - $s\ a \rightarrow_{\texttt{min}} s'\ b\ (c)_{\downarrow_{\sigma'}}$ (where $\sigma' \in K$) can be applied once
  - $s\ a \rightarrow_{\texttt{max}} s'\ b\ (c)_{\downarrow_{\sigma'}}$ (where $\sigma' \in K$) can be applied twice

Faster Byzantine solution

- Each non-faulty process h, $h \in [1, N]$, is modelled by a "process" module, Π_h, which is a combination of $N + 1$ modules:
  - one instance of the "main" module, Ψ_h, which provides the main EIG functionality;
  - plus one instance of the "firewall" communication module, Γ_{hf}, for each process f, $f \in [1, N]$, which takes care of the communication between its "home" process h and its "friend-or-foe" process f.
- Compared to the previous solution, this revised P solution uses fewer cells and rules and only duplex channels.

Faster Byzantine solution

- Each non-faulty process h, $h \in [1, N]$, is modelled by a "process" module, Π_h, which is a combination of $N + 1$ modules:
  - one instance of the "main" module, Ψ_h, which provides the main EIG functionality;
  - plus one instance of the "firewall" communication module, Γ_{hf}, for each process f, $f \in [1, N]$, which takes care of the communication between its "home" process h and its "friend-or-foe" process f.
- Compared to the previous solution, this revised P solution uses fewer cells and rules and only duplex channels.

Faster Byzantine solution

- Each non-faulty process h, $h \in [1, N]$, is modelled by a "process" module, Π_h, which is a combination of $N + 1$ modules:
  - one instance of the "main" module, Ψ_h, which provides the main EIG functionality;
  - plus one instance of the "firewall" communication module, Γ_{hf}, for each process f, $f \in [1, N]$, which takes care of the communication between its "home" process h and its "friend-or-foe" process f.
- Compared to the previous solution, this revised P solution uses fewer cells and rules and only duplex channels.

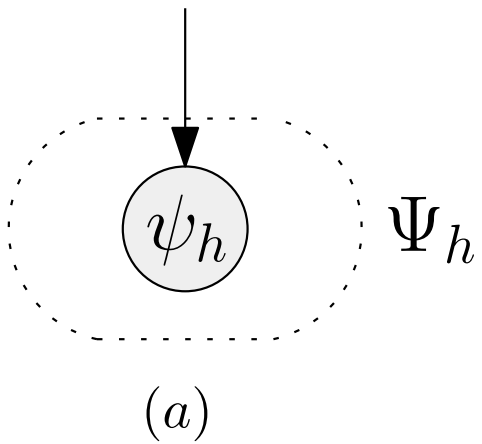
Faster Byzantine solution

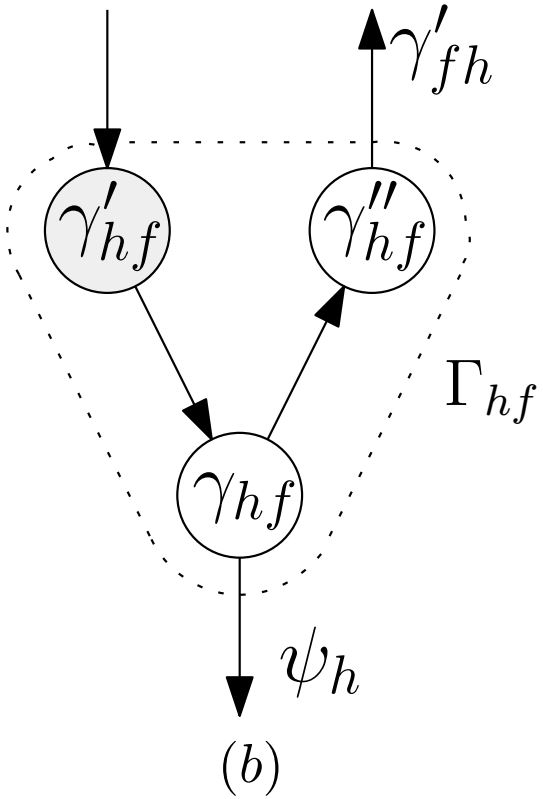
- Each non-faulty process h, $h \in [1, N]$, is modelled by a "process" module, Π_h, which is a combination of $N + 1$ modules:
  - one instance of the "main" module, Ψ_h, which provides the main EIG functionality;
  - plus one instance of the "firewall" communication module, Γ_{hf}, for each process f, $f \in [1, N]$, which takes care of the communication between its "home" process h and its "friend-or-foe" process f.
- Compared to the previous solution, this revised P solution uses fewer cells and rules and only duplex channels.

Faster Byzantine solution

- Each non-faulty process h, $h \in [1, N]$, is modelled by a "process" module, Π_h, which is a combination of $N + 1$ modules:
  - one instance of the "main" module, Ψ_h, which provides the main EIG functionality;
  - plus one instance of the "firewall" communication module, Γ_{hf}, for each process f, $f \in [1, N]$, which takes care of the communication between its "home" process h and its "friend-or-foe" process f.
- Compared to the previous solution, this revised P solution uses fewer cells and rules and only duplex channels.

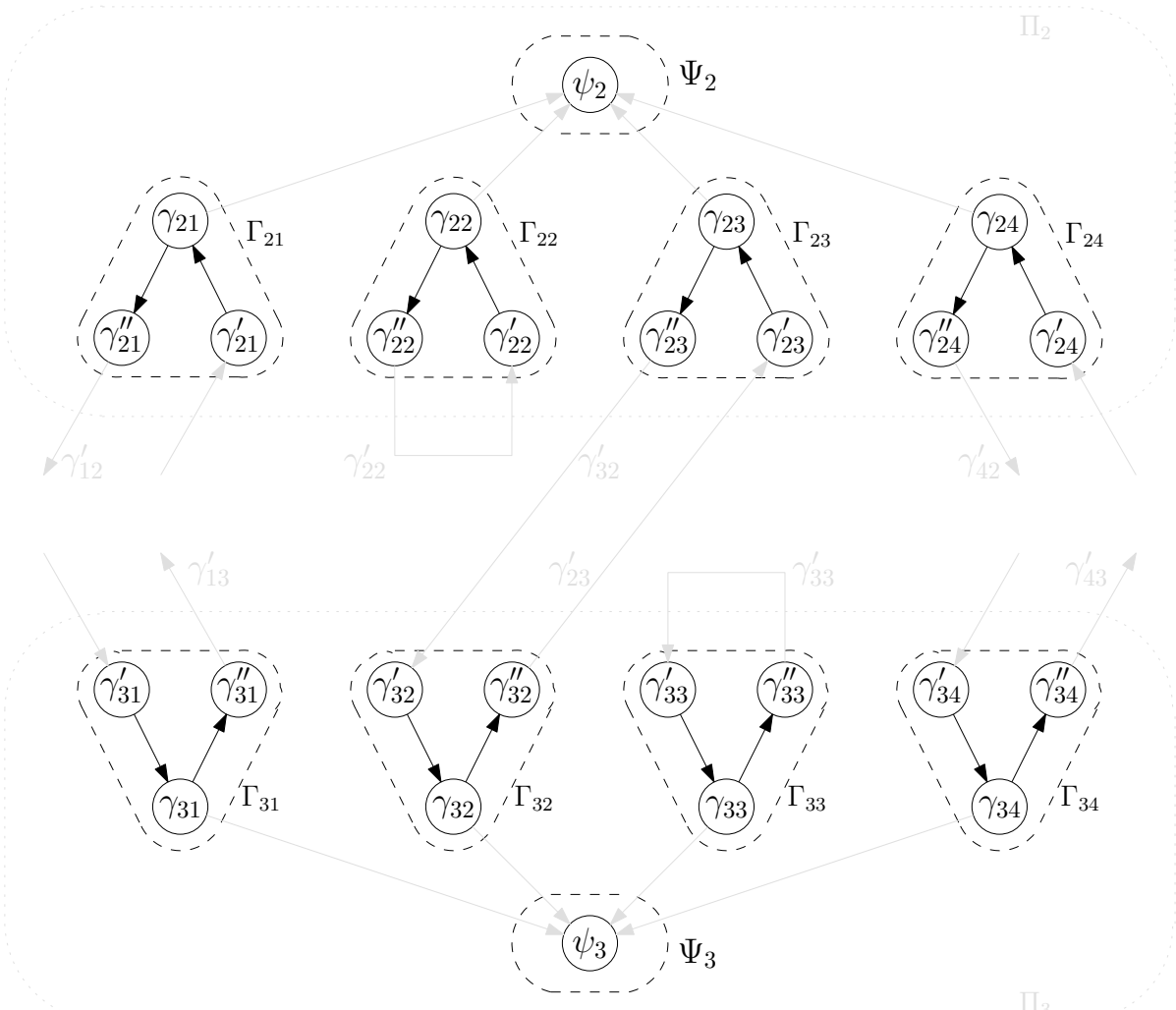
Elementary P modules



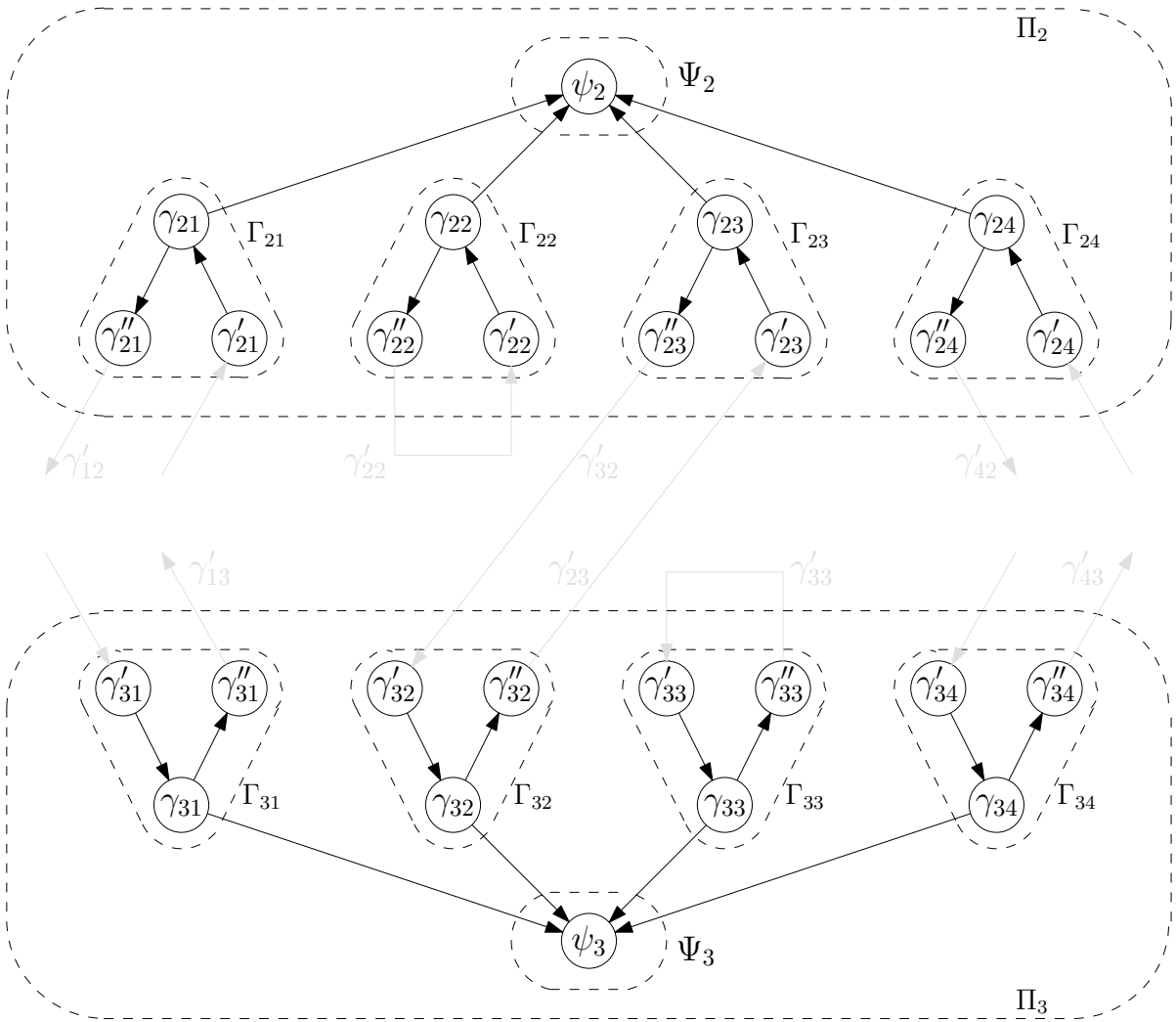


- (a) "main" module (EIG)
- (b) "firewall" module (communication)

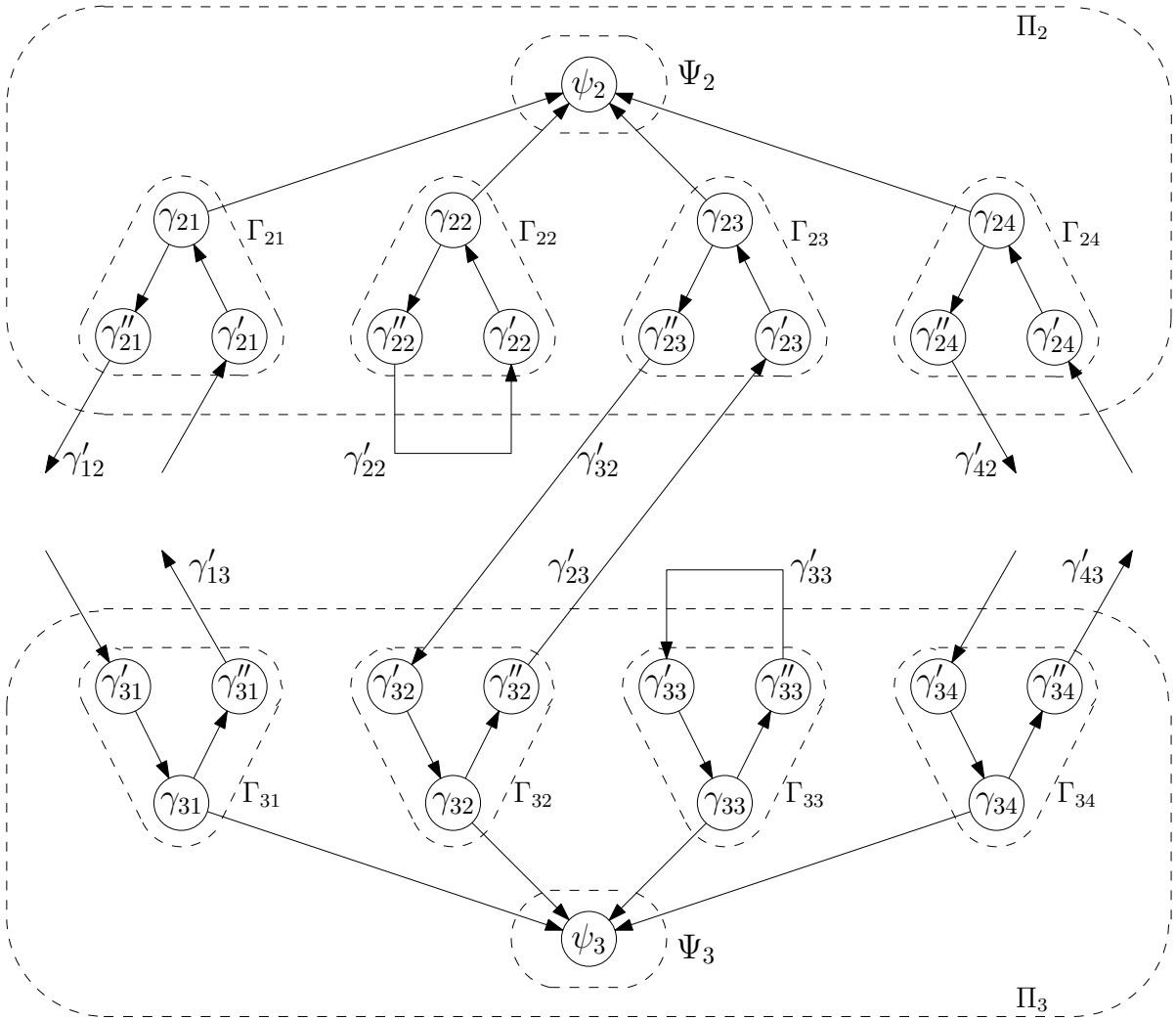
Modular composition of the Byzantine scenario $N = 4$



Modular composition of the Byzantine scenario $N = 4$



Modular composition of the Byzantine scenario $N = 4$



Conclusions

- We proposed an improved generic version of P modules, an extensible framework for recursive composition of P systems,
- and used it to provide a leaner and faster P solution for the Byzantine agreement algorithm, based on EIG trees.
- Our revised P solution uses only duplex channels and fewer cells,
- while improving overall running time from $9L + 6$ to $6L + 1$, where L is the number of messaging rounds.
- Solutions have been extensively verified on our own simulator.

Conclusions

- We proposed an improved generic version of P modules, an extensible framework for recursive composition of P systems,
- and used it to provide a leaner and faster P solution for the Byzantine agreement algorithm, based on EIG trees.
- Our revised P solution uses only duplex channels and fewer cells,
- while improving overall running time from $9L + 6$ to $6L + 1$, where L is the number of messaging rounds.
- Solutions have been extensively verified on our own simulator.

Conclusions

- We proposed an improved generic version of P modules, an extensible framework for recursive composition of P systems,
- and used it to provide a leaner and faster P solution for the Byzantine agreement algorithm, based on EIG trees.
- Our revised P solution uses only duplex channels and fewer cells,
- while improving overall running time from $9L + 6$ to $6L + 1$, where L is the number of messaging rounds.
- Solutions have been extensively verified on our own simulator.

Conclusions

- We proposed an improved generic version of P modules, an extensible framework for recursive composition of P systems,
- and used it to provide a leaner and faster P solution for the Byzantine agreement algorithm, based on EIG trees.
- Our revised P solution uses only duplex channels and fewer cells,
- while improving overall running time from $9L + 6$ to $6L + 1$, where L is the number of messaging rounds.
- Solutions have been extensively verified on our own simulator.

Conclusions

- We proposed an improved generic version of P modules, an extensible framework for recursive composition of P systems,
- and used it to provide a leaner and faster P solution for the Byzantine agreement algorithm, based on EIG trees.
- Our revised P solution uses only duplex channels and fewer cells,
- while improving overall running time from $9L + 6$ to $6L + 1$, where L is the number of messaging rounds.
- Solutions have been extensively verified on our own simulator.

Conclusions

- We proposed an improved generic version of P modules, an extensible framework for recursive composition of P systems,
- and used it to provide a leaner and faster P solution for the Byzantine agreement algorithm, based on EIG trees.
- Our revised P solution uses only duplex channels and fewer cells,
- while improving overall running time from $9L + 6$ to $6L + 1$, where L is the number of messaging rounds.
- Solutions have been extensively verified on our own simulator.

Solved open problem

- Since this paper was accepted, we indirectly answered one of the mentioned open problems.
- We can extend our P system solution to cover $2F + 1$ connected graphs, but not necessarily complete.
- Our MeCBIC paper offers efficient P system solutions to two fundamental problems in graph theory: finding the maximum number of node- and edge-disjoint paths between a source node and target node.

Solved open problem

- Since this paper was accepted, we indirectly answered one of the mentioned open problems.
- We can extend our P system solution to cover $2F + 1$ connected graphs, but not necessarily complete.
- Our MeCBIC paper offers efficient P system solutions to two fundamental problems in graph theory: finding the maximum number of node- and edge-disjoint paths between a source node and target node.

Solved open problem

- Since this paper was accepted, we indirectly answered one of the mentioned open problems.
- We can extend our P system solution to cover $2F + 1$ connected graphs, but not necessarily complete.
- Our MeCBIC paper offers efficient P system solutions to two fundamental problems in graph theory: finding the maximum number of node- and edge-disjoint paths between a source node and target node.

Solved open problem

- Since this paper was accepted, we indirectly answered one of the mentioned open problems.
- We can extend our P system solution to cover $2F + 1$ connected graphs, but not necessarily complete.
- Our MeCBIC paper offers efficient P system solutions to two fundamental problems in graph theory: finding the maximum number of node- and edge-disjoint paths between a source node and target node.

Open problems

- Is it possible to solve the Byzantine agreement problem using fewer or even a fixed number of resources (objects, rules)?
- Is it possible to offer an efficient solution with a better message complexity (currently exponential)?
- Is it possible to provide a "native" P system version, where each process is modelled by just one cell (without "firewalls")?
- Is it possible to extend the P module framework to support timing requirements and interactivity (possibly endless)?
- What extensions would be required and how would they fit into the P systems framework?

Open problems

- Is it possible to solve the Byzantine agreement problem using fewer or even a fixed number of resources (objects, rules)?
- Is it possible to offer an efficient solution with a better message complexity (currently exponential)?
- Is it possible to provide a "native" P system version, where each process is modelled by just one cell (without "firewalls")?
- Is it possible to extend the P module framework to support timing requirements and interactivity (possibly endless)?
- What extensions would be required and how would they fit into the P systems framework?

Open problems

- Is it possible to solve the Byzantine agreement problem using fewer or even a fixed number of resources (objects, rules)?
- Is it possible to offer an efficient solution with a better message complexity (currently exponential)?
- Is it possible to provide a "native" P system version, where each process is modelled by just one cell (without "firewalls")?
- Is it possible to extend the P module framework to support timing requirements and interactivity (possibly endless)?
- What extensions would be required and how would they fit into the P systems framework?

Open problems

- Is it possible to solve the Byzantine agreement problem using fewer or even a fixed number of resources (objects, rules)?
- Is it possible to offer an efficient solution with a better message complexity (currently exponential)?
- Is it possible to provide a "native" P system version, where each process is modelled by just one cell (without "firewalls")?
- Is it possible to extend the P module framework to support timing requirements and interactivity (possibly endless)?
- What extensions would be required and how would they fit into the P systems framework?

Open problems

- Is it possible to solve the Byzantine agreement problem using fewer or even a fixed number of resources (objects, rules)?
- Is it possible to offer an efficient solution with a better message complexity (currently exponential)?
- Is it possible to provide a "native" P system version, where each process is modelled by just one cell (without "firewalls")?
- Is it possible to extend the P module framework to support timing requirements and interactivity (possibly endless)?
- What extensions would be required and how would they fit into the P systems framework?

Open problems

- Is it possible to solve the Byzantine agreement problem using fewer or even a fixed number of resources (objects, rules)?
- Is it possible to offer an efficient solution with a better message complexity (currently exponential)?
- Is it possible to provide a "native" P system version, where each process is modelled by just one cell (without "firewalls")?
- Is it possible to extend the P module framework to support timing requirements and interactivity (possibly endless)?
- What extensions would be required and how would they fit into the P systems framework?

Thank you

Thank you for your attention!