Formally Proving and Enhancing a Self-Stabilising Distributed Algorithm

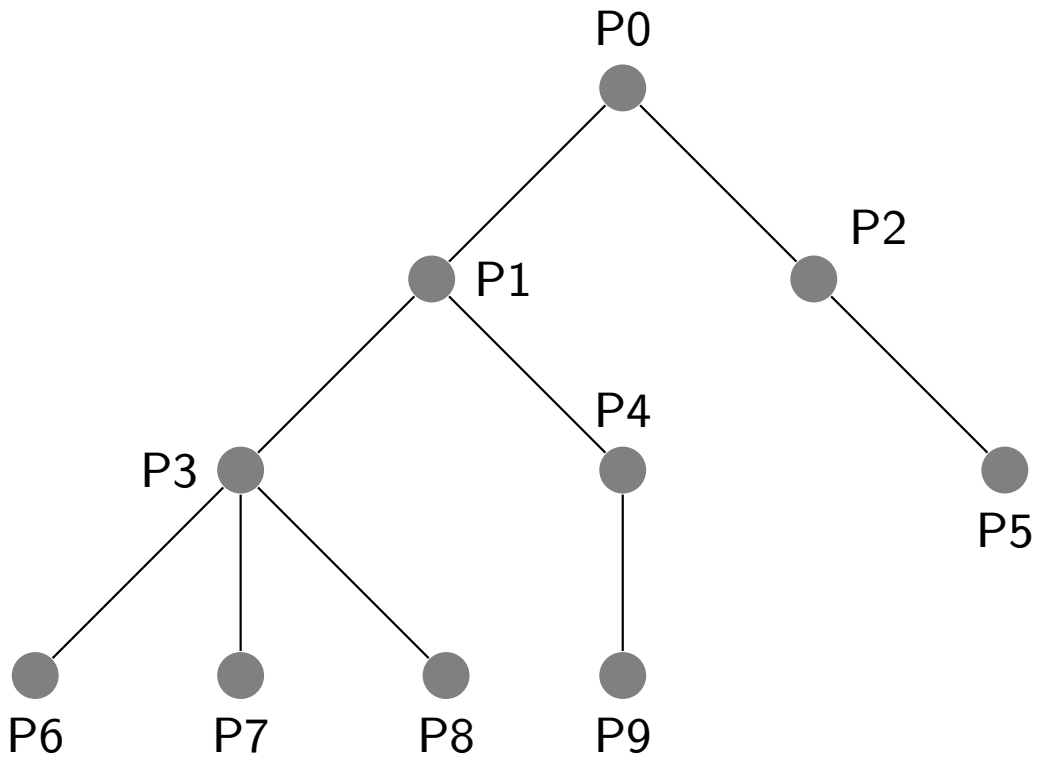
Camille Coti ♣ , Charles Lakos ♠ , Laure Petrucci ♣

♣ LIPN, CNRS UMR 7030, SPC, Université Paris 13, France
♠ Department of Computer Science, University of Adelaide, Australia

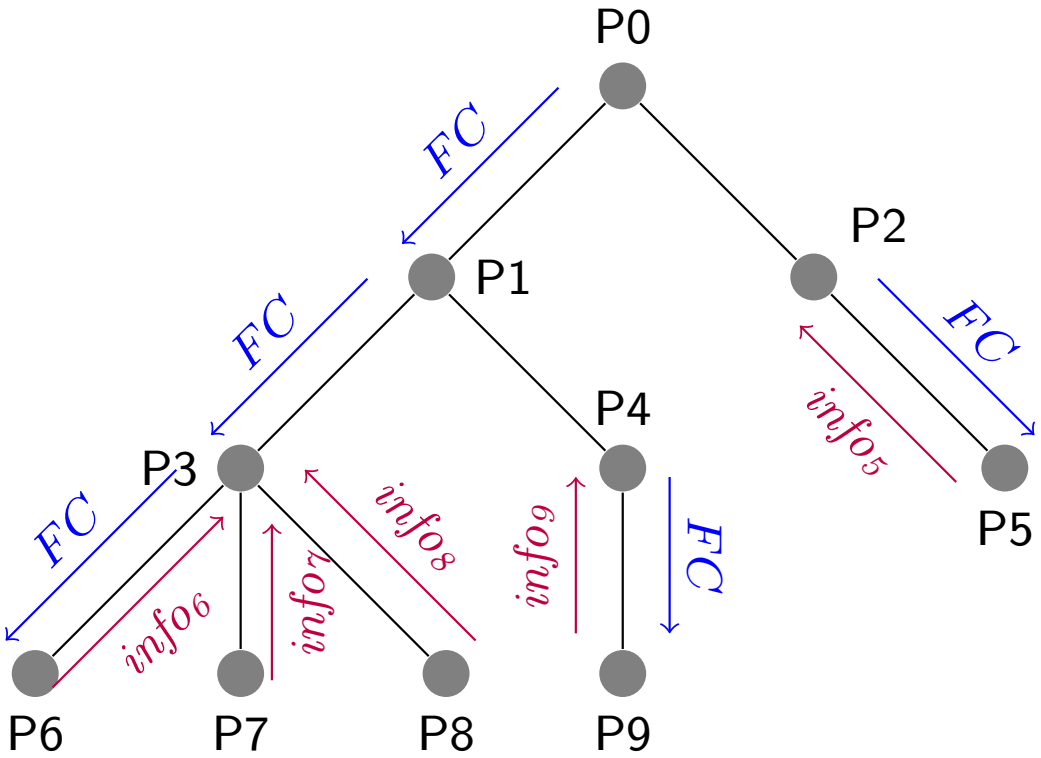
PNSE
June 20th, 2016

Roadmap

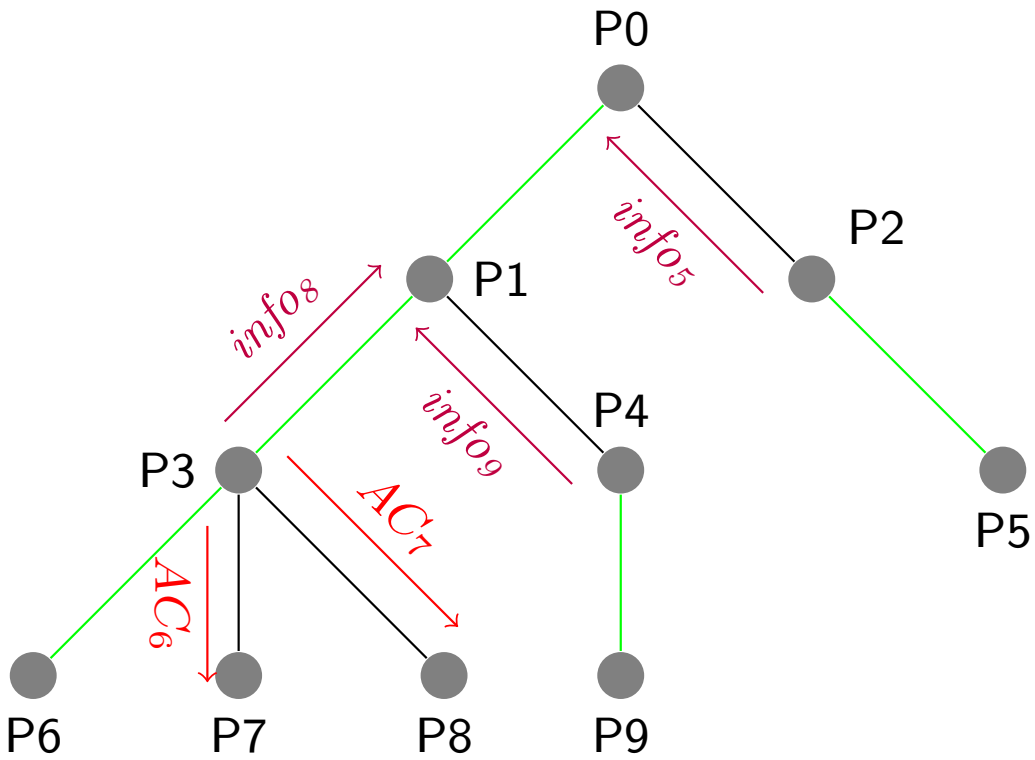
The algorithm



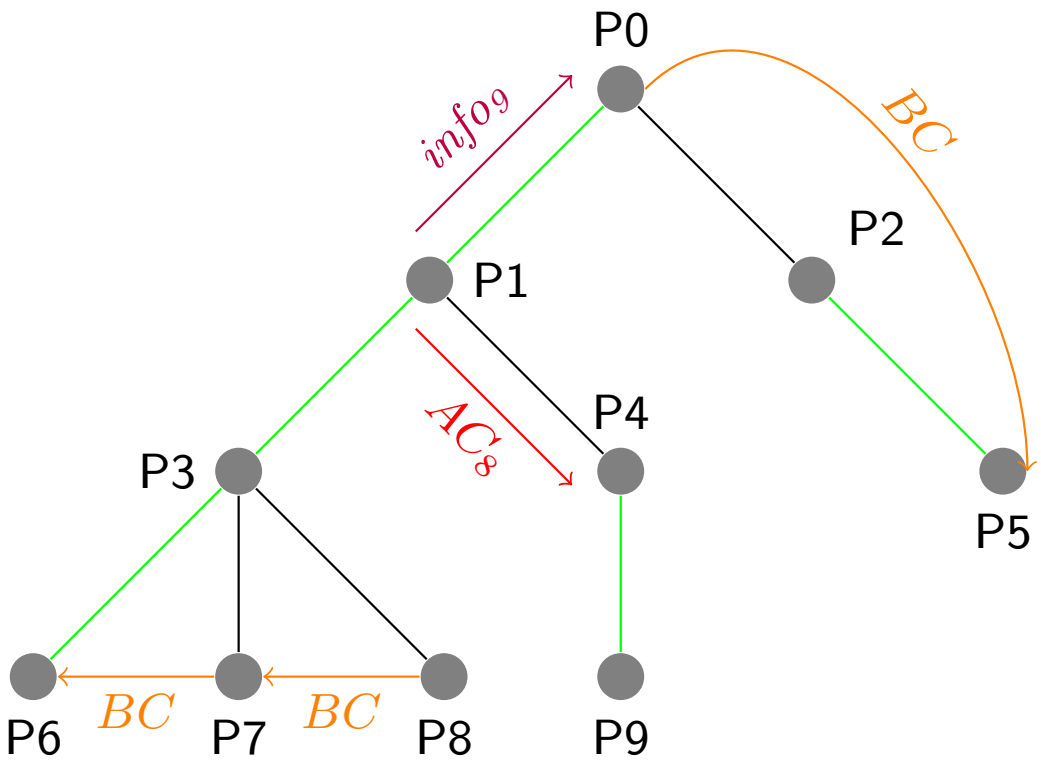
The algorithm



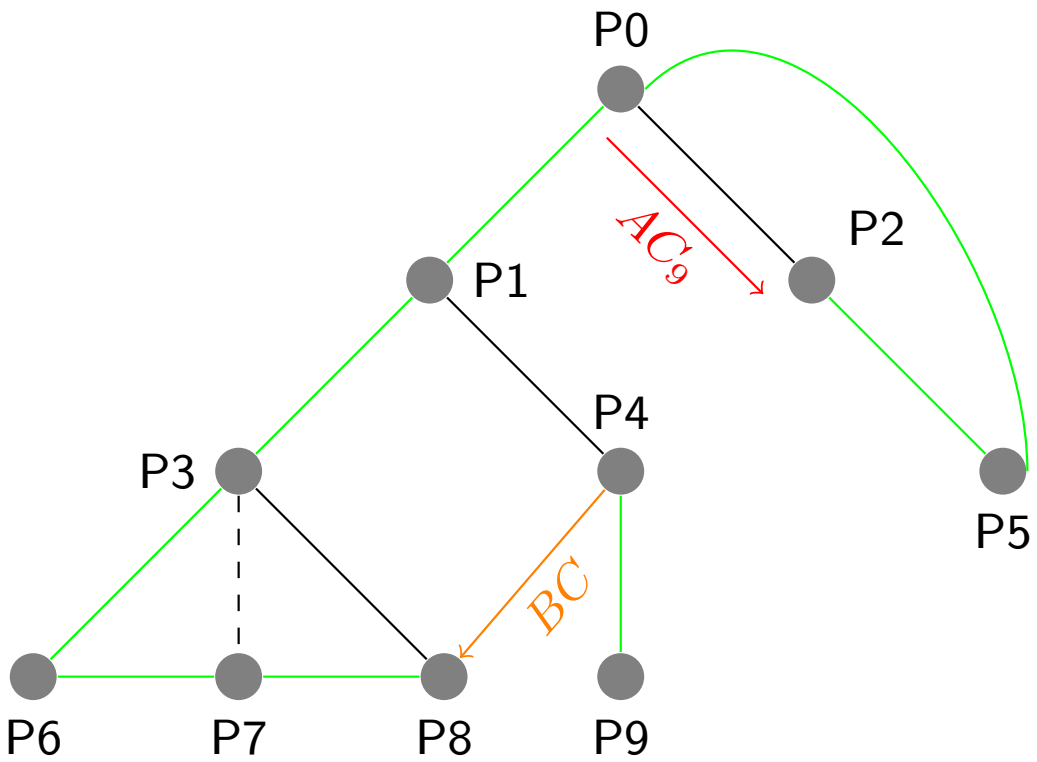
The algorithm



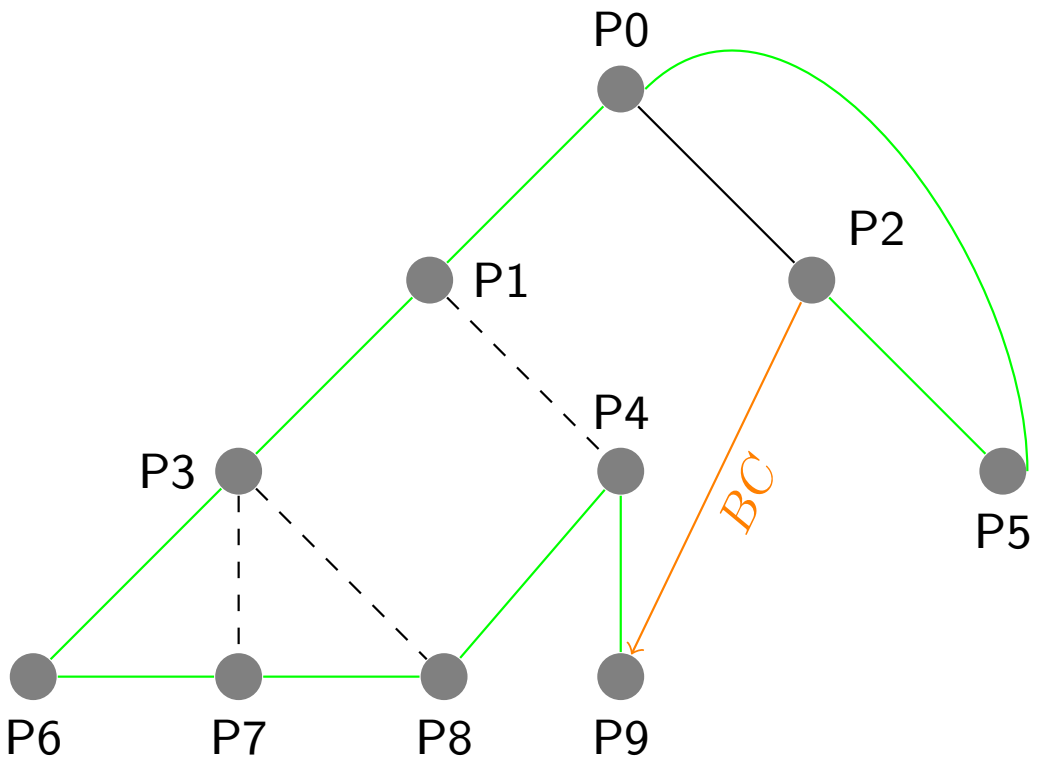
The algorithm



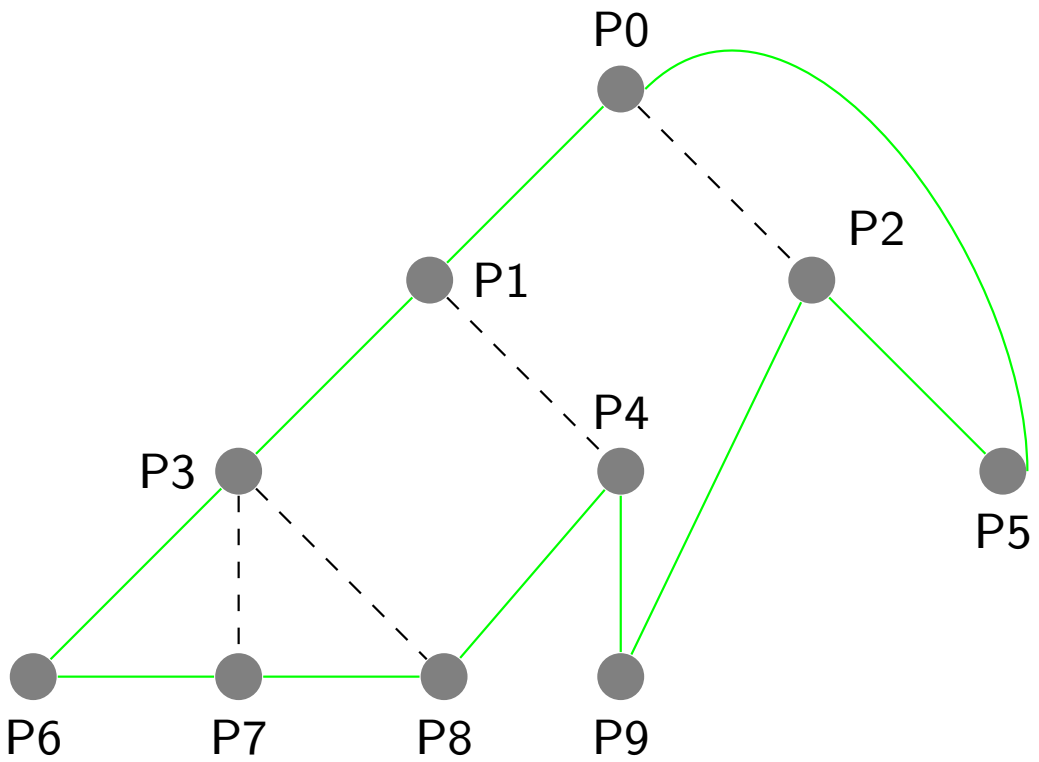
The algorithm



The algorithm



The algorithm



Properties of the algorithm

Transforms a tree topology to a ring

- Self-stabilising
- From *any* initial configuration (*ie* any initial tree), reaches a *legitimate* configuration
  - Initial configuration: tree, result of a process spawning
  - Legitimate configuration: ring
- When a legitimate configuration is reached, the system stays in this configuration
- Silent: after the system has stabilised, the communications are *fixed*

Properties of the algorithm

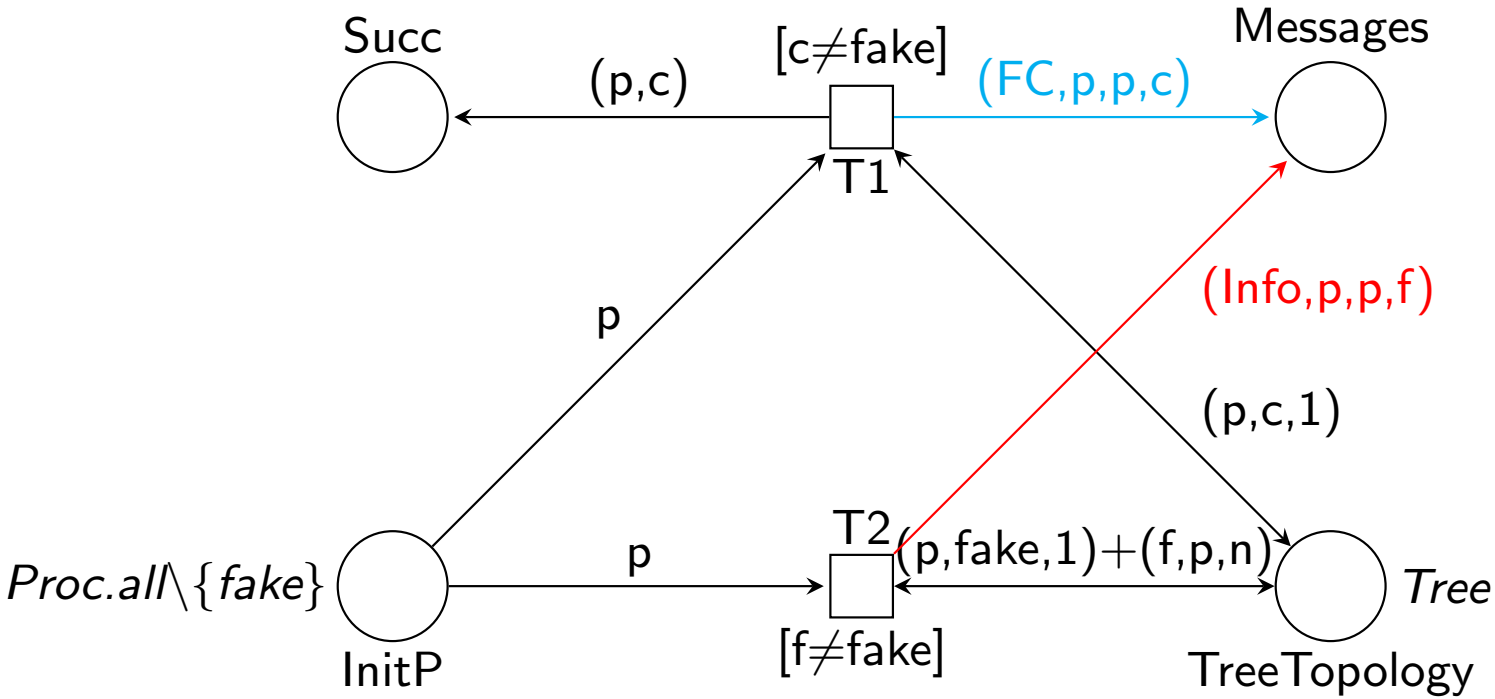
Transforms a tree topology to a ring

- Self-stabilising
- From *any* initial configuration (*ie* any initial tree), reaches a *legitimate* configuration
  - Initial configuration: tree, result of a process spawning
  - Legitimate configuration: ring
- When a legitimate configuration is reached, the system stays in this configuration
- Silent: after the system has stabilised, the communications are *fixed*

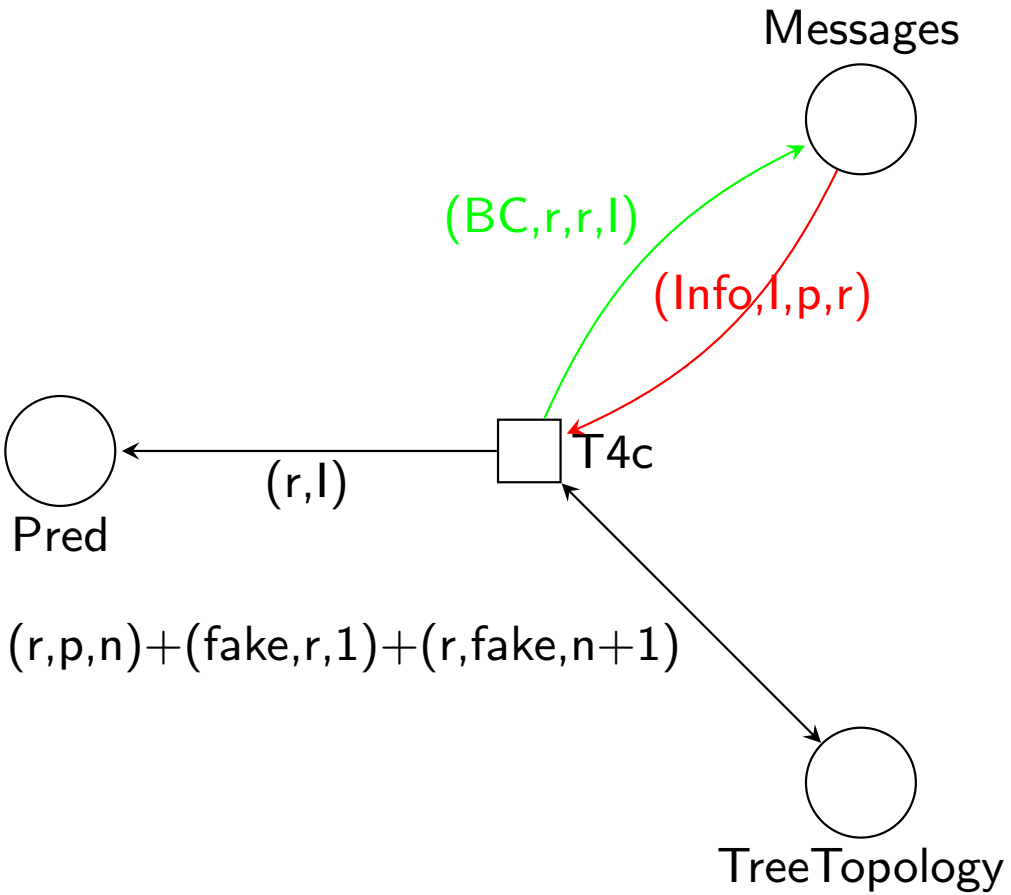
First step of the construction of a **Binomial Graph**

- Many interesting properties, making it very suitable for overlay networks supporting the run-time environment of distributed applications
- The ring-to-BMG construction is trivially self-stabilising
- Combination of two self-stabilising algorithms = self-stabilising

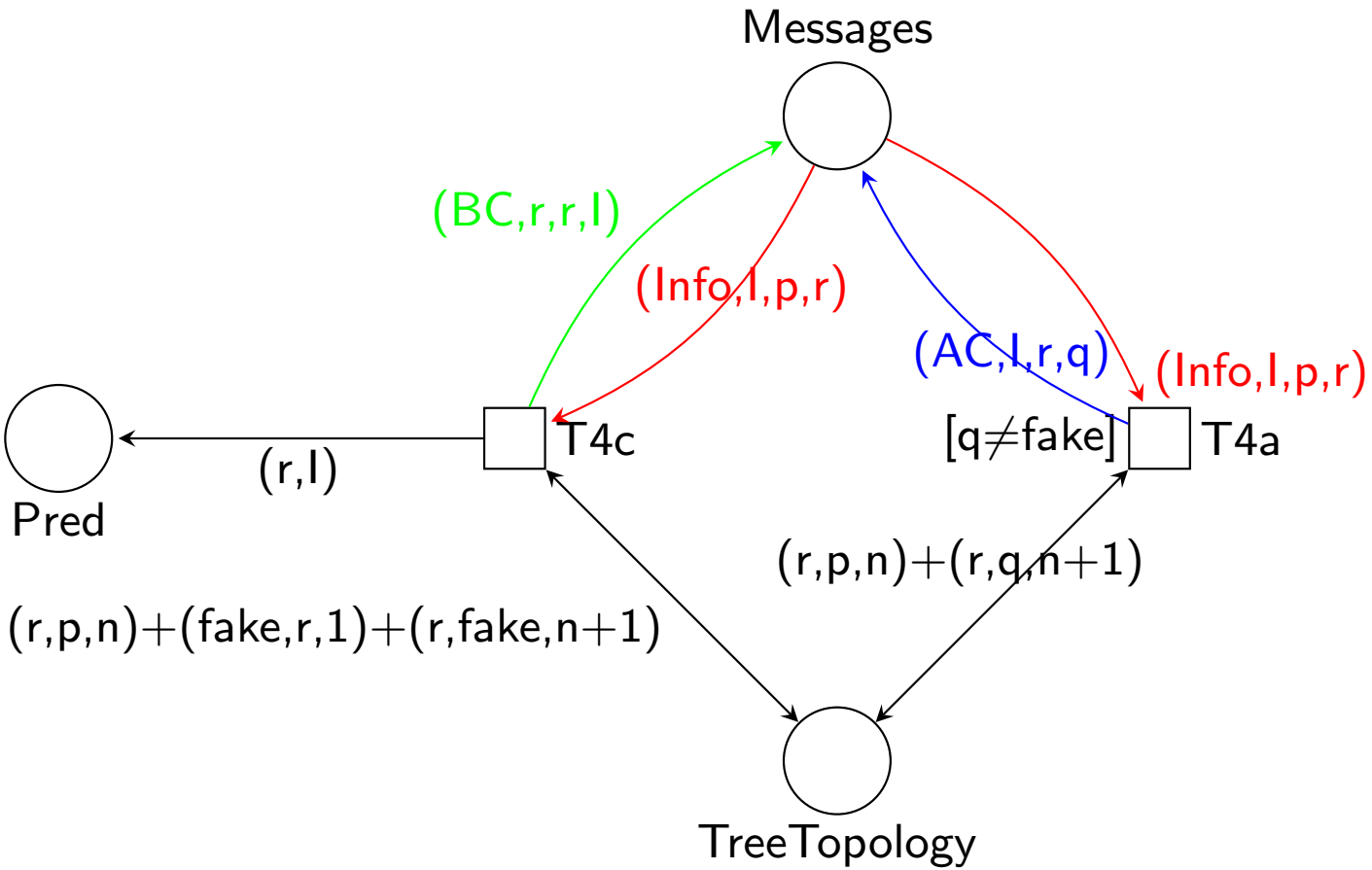
Coloured Petri Net Model: Intialisation



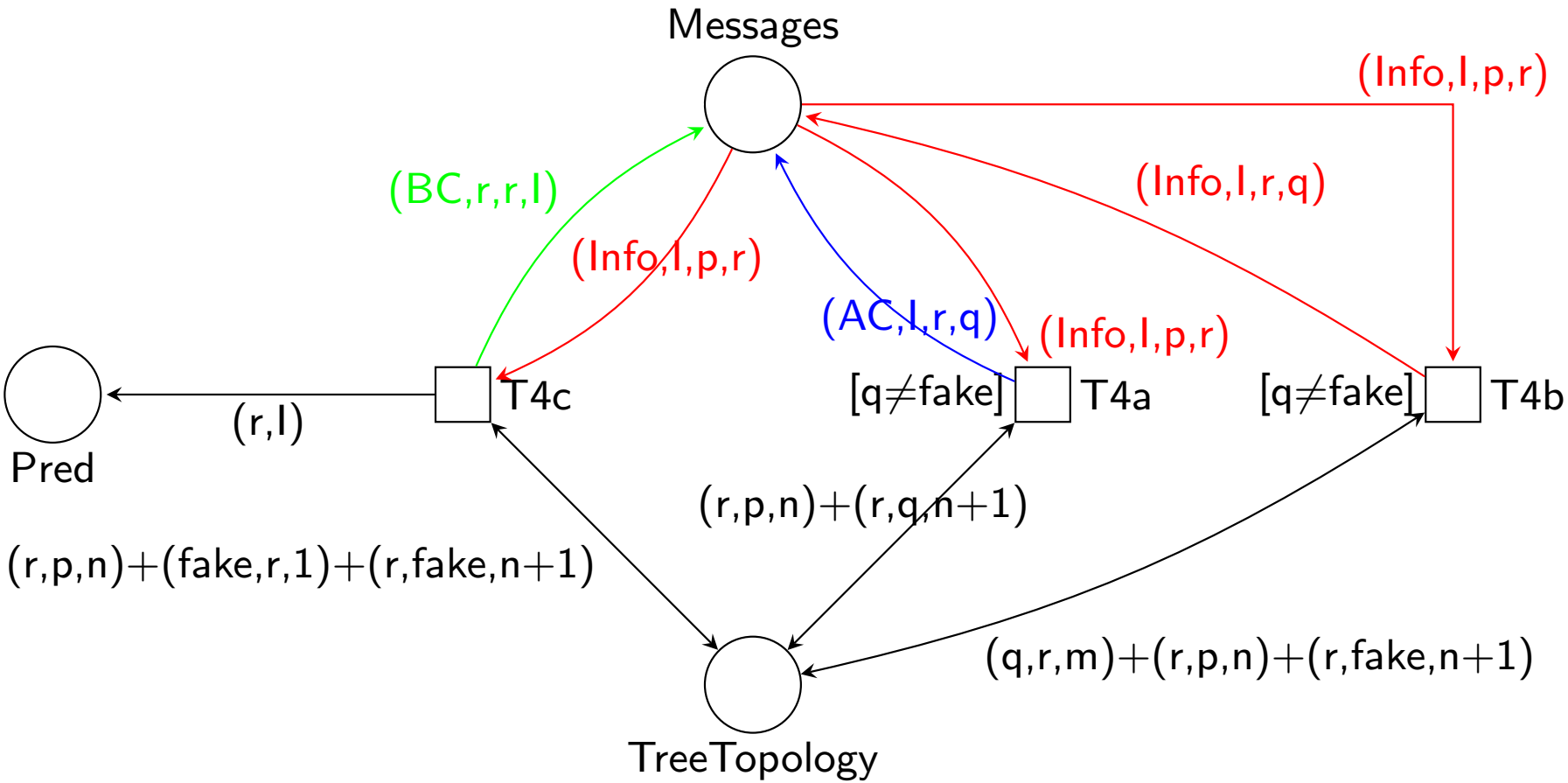
Coloured Petri Net Model: Main Part



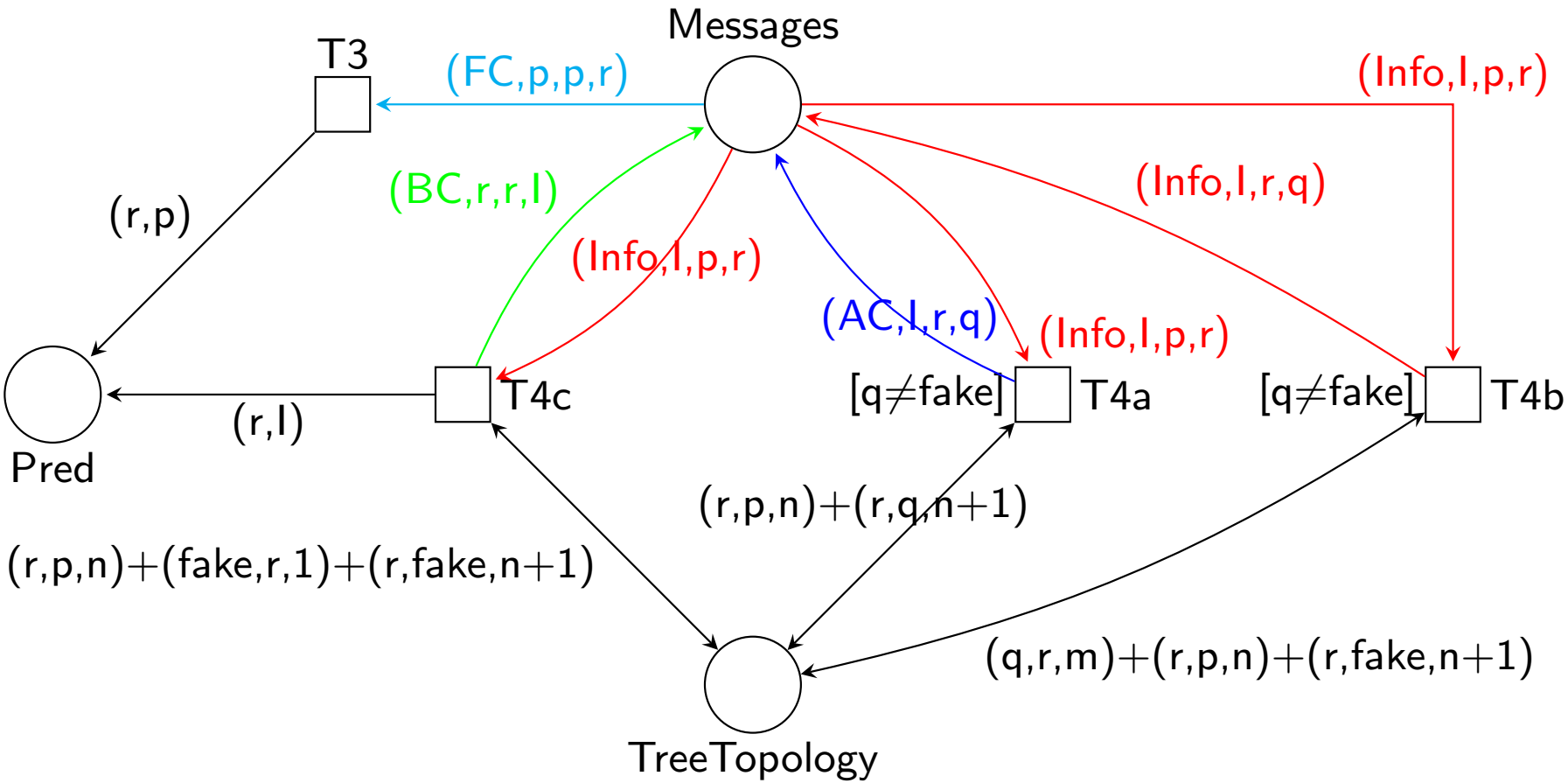
Coloured Petri Net Model: Main Part



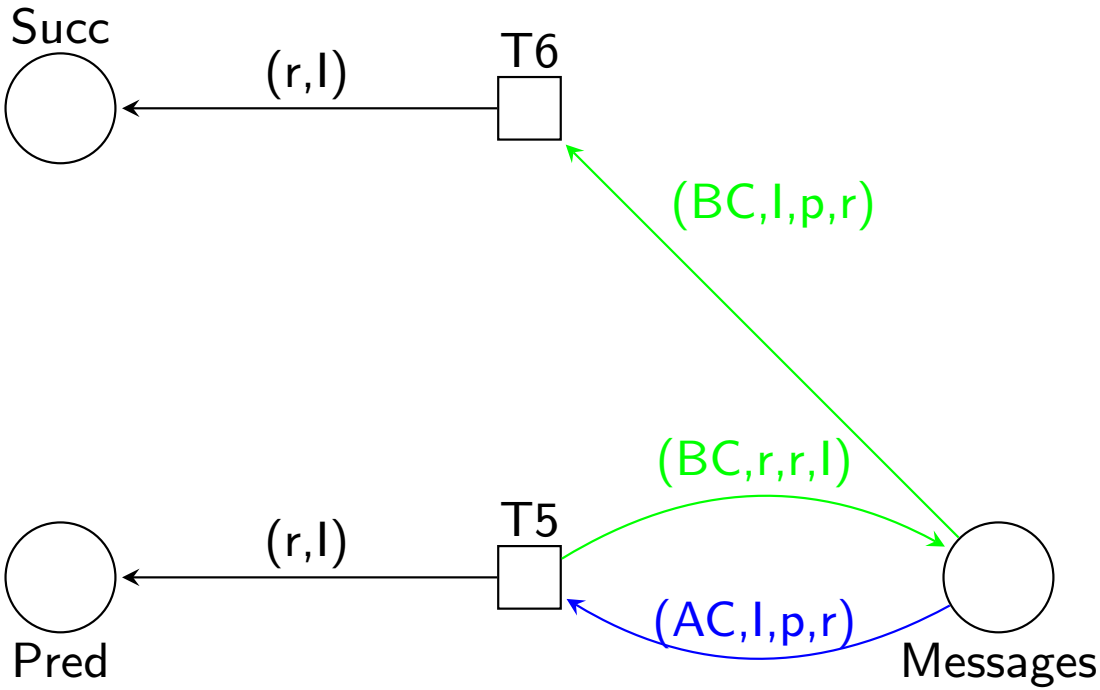
Coloured Petri Net Model: Main Part



Coloured Petri Net Model: Main Part



Coloured Petri Net Model: End



Formal Analysis of the Algorithm Properties

Invariants

- $\texttt{InitP} + \pi_1(\texttt{Succ}) + \pi_2(Messages(\texttt{Info})) + \pi_2(Messages(\texttt{AC})) + \pi_4(Messages(\texttt{BC})) = \texttt{Proc} \setminus \{\texttt{fake}\}$
- $\texttt{Succ} + \pi_{4,2}(Messages(\texttt{BC})) = \pi_{3,4}(Messages(\texttt{FC})) + \pi_{2,1}(\texttt{Pred})$

Formal Analysis of the Algorithm Properties

Invariants

- $\texttt{InitP} + \pi_1(\texttt{Succ}) + \pi_2(Messages(\texttt{Info})) + \pi_2(Messages(\texttt{AC})) + \pi_4(Messages(\texttt{BC})) = \texttt{Proc} \setminus \{\texttt{fake}\}$
- $\texttt{Succ} + \pi_{4,2}(Messages(\texttt{BC})) = \pi_{3,4}(Messages(\texttt{FC})) + \pi_{2,1}(\texttt{Pred})$

Liveness properties

- For any tree with at least two nodes, either transition `T1` or transition `T2` can fire for every node. Thus, `InitP` will eventually be empty.
- All messages can eventually be uniquely consumed. (silentness)

Formal Analysis of the Algorithm Properties

Invariants

- $\texttt{InitP} + \pi_1(\texttt{Succ}) + \pi_2(Messages(\texttt{Info})) + \pi_2(Messages(\texttt{AC})) + \pi_4(Messages(\texttt{BC})) = \texttt{Proc} \setminus \{\texttt{fake}\}$
- $\texttt{Succ} + \pi_{4,2}(Messages(\texttt{BC})) = \pi_{3,4}(Messages(\texttt{FC})) + \pi_{2,1}(\texttt{Pred})$

Liveness properties

- For any tree with at least two nodes, either transition `T1` or transition `T2` can fire for every node. Thus, `InitP` will eventually be empty.
- All messages can eventually be uniquely consumed. (silentness)

Termination

The algorithm terminates and is self-stabilising.

Improving the Model to Improve the Algorithm

$Pred$ and $Succ$ are mirror images

Only one of them is necessary:

- Remove $Pred$ and FC messages (better when a lot of leaves in the tree)
- or remove $Succ$ and BC messages

Conclusion & Perspectives

We have:

- modelled an algorithm transforming a tree topology into a ring
- provided an elegant proof of self-stabilisation, plus additional properties
- derived simplified algorithms

Conclusion & Perspectives

We have:

- modelled an algorithm transforming a tree topology into a ring
- provided an elegant proof of self-stabilisation, plus additional properties
- derived simplified algorithms

Further work will include:

- generalising the approach to self-stabilisation of other protocols
- designing a *self-healing algorithm*, based on the expertise gained

Self-stabilisation

Technique to **tolerate failures**

- From **any** initial configuration, reach a legitimate configuration
- Example:
  1. Start from a tree
  2. Build a ring
  3. A process fails $\rightarrow$ the ring is broken, equiv. to a degenerated tree
  4. A new process may or may not be respawned: we still have a (degenerated) tree
  5. Run the algorithm again to build a new ring

Difference with **self-healing**

- Self-stabilisation: restart the algorithm starting from the new configuration
- Self-healing: run a local protocol to recover from the failure
  - *e.g. re-knit the ring*

Self-stabilisation

Property: self-stabilising algorithms are **silent**

- Once the algorithm has stabilised, the communications are **fixed**
  - The processes can always send the same message to their neighbours
  - ... or no message at all.

Example: Bellman-Ford (used in RIP and BGP dynamic routing protocols)

- Builds a collector tree on each router of the network
  - Eventually each router knows the shortest path to the other neworks
- Communicate the length of the shortest path to each (known) network
- Once the system has stabilised:
  - Each router sends the length of the shortest paths to the networks it kows
  - Its neighbours do not make iny chage because they already know the shortest paths to these networks
  - Nothing changes
  - The system stays in the same configuration and the communications are fixed.

Failure detection

- Another problem on its own.
- Was proved impossible with asynchronous communications.
- In general: assume that we have an *eventually perfect failure* detector.