Revisited Convergence of a Self-Stabilizing BFS Spanning Tree Algorithm

Karine Altisen, Marius Bozga

Pavedys meeting – 13 mai 2025

UGA Université Grenoble Alpes

Verimag

UNIVERSITÉ DE GENÈVE

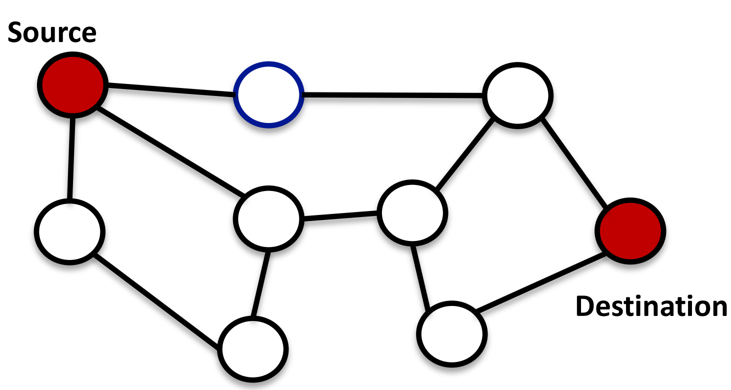
Distributed Algorithms

- **Autonomous Computing Entities**
  Local Memory, Local Program
  Communication functions

- **Communication Links**

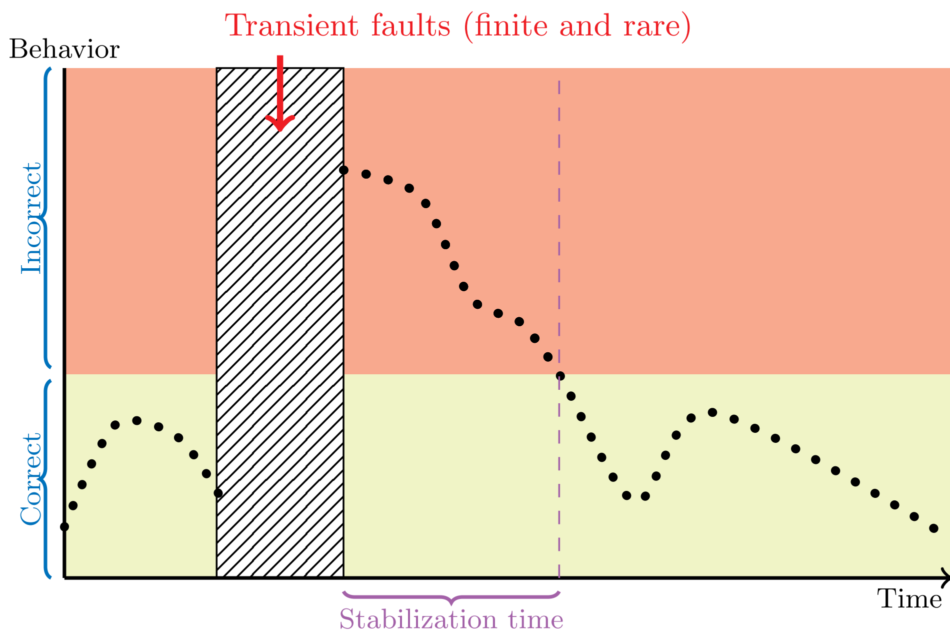
➔ Solve a common problem
despite Distribution
Asynchrony
… and Faults

e.g. Routing data from a process to another



Self-Stabilization

Non-Masking Fault Tolerance



Computational Model = Atomic State Model

- Locally shared memory model
- Local algorithm = list of guarded actions

communication abstraction

Configuration = the state of all processes

Atomic Step

- Each process reads its local and neighbor's variables

→ Enabled?

- daemon selection → Set of processes to be executed

asynchrony modeling

- Each selected process → Updates its local variables

Execution = stream of configurations

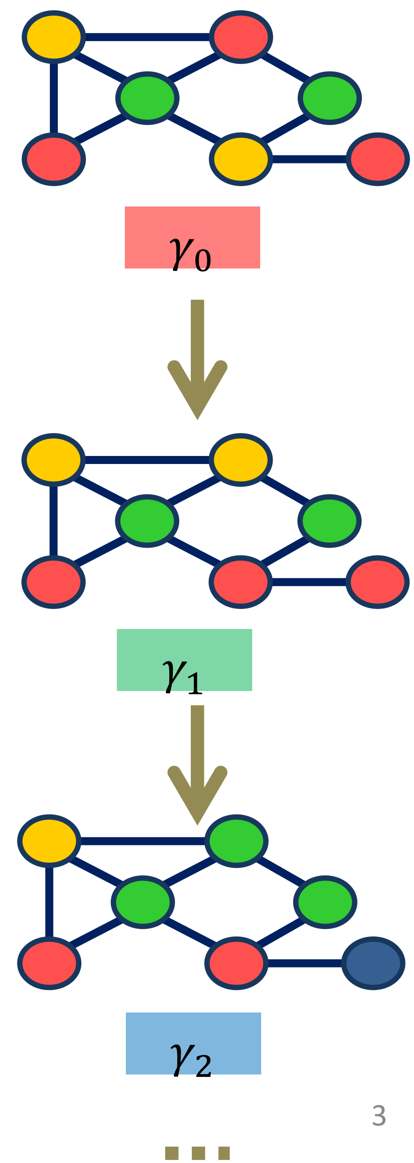
Linked by atomic steps →

Maximal = finite iff the last config. is terminal

(= no more node enabled)

Daemon = *localization* and *fairness* properties

HERE: Unfair Daemon = no constraint



Self-Stabilizing BFS Spanning Tree

Dolev - Israeli - Moran - 1993

Bidir rooted network

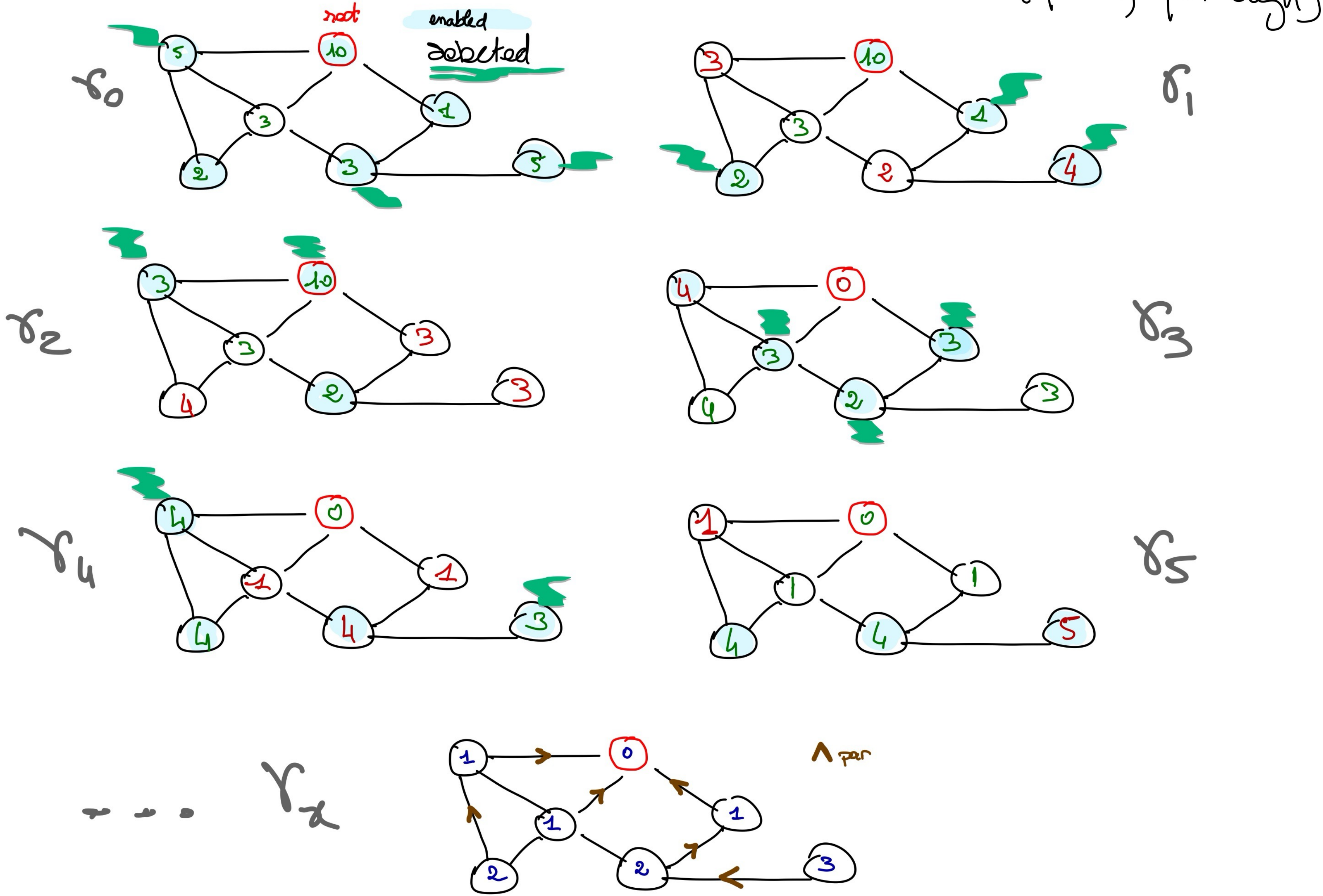
Algorithm 1 Algorithm BFS, code for each node p.

Constant Local Input: $p.neigh \subseteq Node$; $p.root \in \{t, f\}$
Local Variables: $p.d \in \mathbb{N}$; $p.par \in Node$
Macros:
$Dist_p = \min\{q.d + 1, q \in p.neigh\}$
$Par_{dist} = \text{fst } \{q \in p.neigh, q.d + 1 = p.d\}$
Algorithm for the root(`p.root = true`)
`Root` Action: if $p.d \neq 0$ then
$p.d$ is set to 0
Algorithm for any non-root node(`p.root = false`)
CD Action: if $p.d \neq Dist_p$ then
$p.d$ is set to $Dist_p$
CP Action: if $p.d = Dist_p$ and $p.par.d + 1 \neq p.d$ then
$p.par$ is set to Par_{dist}

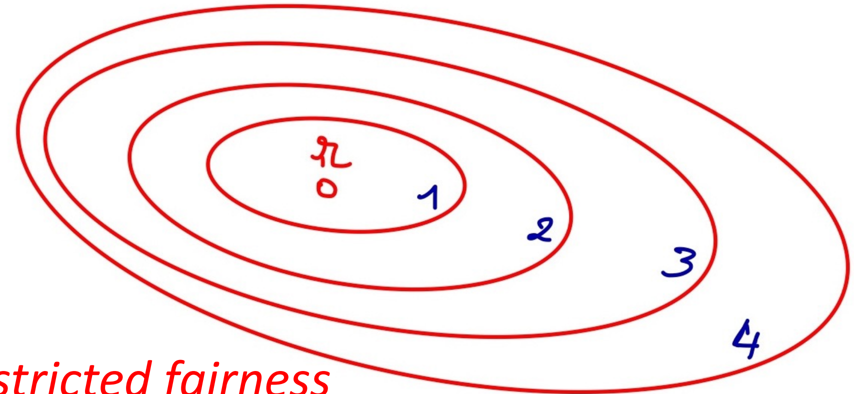
Example of execution

root: $p.d = 0$

non.root: $p.d = \min\{q.d+1, q \in Neigh\}$



Context and Contributions

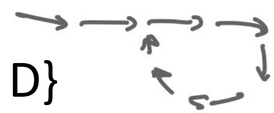


Existing proofs for this algorithm

- Initial proof
- Machine checked proof (PADEC/Coq)

→ *under restricted fairness*

Assume a cyclic execution

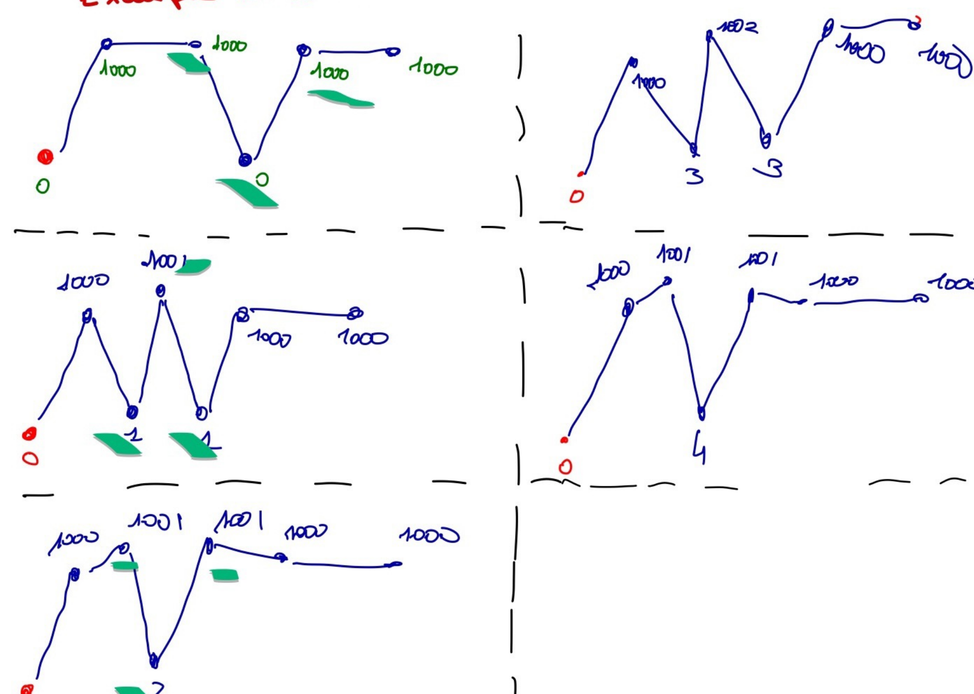


and prove FALSE

- Unfair daemon, bounded variables d in {0, …, D}

→ proof by contradiction

Example on a line



Our Contribution

- Unfair daemon
- Unbounded variables d in N
- Constructive and machine-checked proof using PADEC/Coq

PADEC: the PADEC Project

«Preuves d'Algorithmes Distribués En Coq»

"Proofs of Distributed Algorithms using Coq/Rocq"

- **Goal:** Formal proofs for *self-stabilizing* algorithms
  in the *Atomic-state Model (ASM)*
  using *Coq/Rocq* and its libraries

- **Challenges: handle every kind of proofs for self-stabilization in the ASM**
  → Correctness and convergence
  → Quantitative properties
  → Time complexity

PADEC provides a Coq/Roq library including:

- General tools
- Computational model and specifications
- Lemmas corresponding to common proof patterns
- Case-studies

Definitions

Examples

PADEC – Big Picture

Common proof patterns & results

Case studies

***Libraries**:*
Setoid support
Streams, LTL,
Counting

Computational Model
ASM Semantics

Assumptions
- **Daemons**
- **Networks**

Unfair, weakly fair,
synchronous

Connected, ring, tree
Identified, (semi-)anonymous
Measures (distance, diameter)

Relational <-> Functional

Specification
- **Self-Stabilization**
- **Problem**
- **Complexity: Steps, Rounds**

BFS spanning tree
Token circulation
Dominating set, Clustering

Induction Schema

BFS spanning tree (rounds)
Dijkstra Token Ring (steps)

Composition
- Hierarchical Collateral

Proof of
- Specification
- Complexity

BFS + KClustering

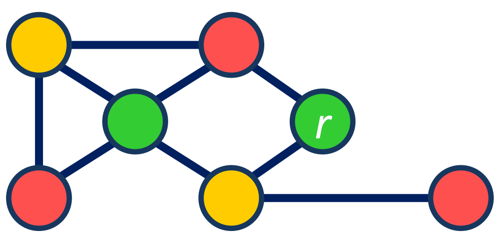
Tools for convergence
Lexico, Well-founded,
Potential & multisets

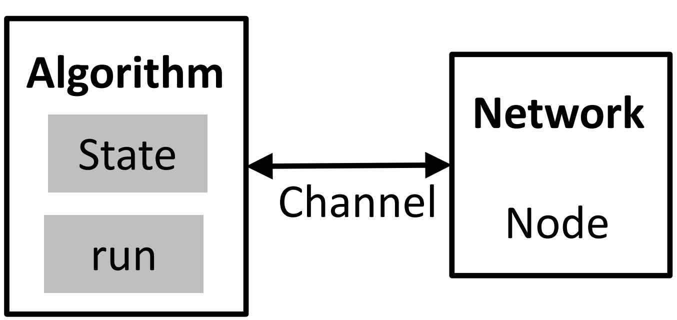
KDomSet, KClustering,
BFS

PADEC: Short How To

Algorithm 1 Algorithm BFS, code for each node p.

Constant Local Input: $p.neigh \subseteq Node$; $p.root \in \{t, f\}$
Local Variables: $p.d \in \mathbb{N}$; $p.par \in Node$
Macros:
$Dist_p = \min\{q.d + 1, q \in p.neigh\}$
$Par_{dist} = \text{fst } \{q \in p.neigh, q.d + 1 = p.d\}$
Algorithm for the root(`p.root = true`)
`Root` Action: if $p.d \neq 0$ then
$p.d$ is set to 0
Algorithm for any non-root node(`p.root = false`)
CD Action: if $p.d \neq Dist_p$ then
$p.d$ is set to $Dist_p$
CP Action: if $p.d = Dist_p$ and $p.par.d + 1 \neq p.d$ then
$p.par$ is set to Par_{dist}





Instantiate ***Algorithm****:*
- State = a record of local var.
- run = a faithful translation

Express ***Assumption****:*
- **Daemon** *e.g., unfair*
- **Network**, *e.g. rooted, bidir, connected*

Express ***Specification****:*
- **Self-stabilizing** w.r.t. a **problem** *e.g., BFS spanning tree*

Prove it!!

Some Elements of Formalization in PADEC

Configuration γ of type Env := Node -> State

Atomic Step : relation Step := Env -> Env -> Prop

Execution e of type Exec := Stream Env such as predicate is_exec: Exec -> Prop

Self-Stabilization : A is self-stabilizing under (**Assume**, **Daemon**) wrt **SPEC** :=
exists **LC** a set of *legitimate* configurations s.t.
forall execution *e* of **A** under (**Assume**, **Daemon**)

TODO

- <u>Closure</u>: if *e* starts **LC** in then *e* remains in **LC**
- <u>Convergence</u>: *e* eventually reaches **LC**
- <u>Specification</u>: if *e* starts in **LC** then *e* satisfies **SPEC**

Already Done

Algorithm 1 Algorithm BFS, code for each node p.

Constant Local Input: $p.neigh \subseteq Node$; $p.root \in \{t, f\}$
Local Variables: $p.d \in \mathbb{N}$; $p.par \in Node$
Macros:
$Dist_p = \min\{q.d + 1, q \in p.neigh\}$
$Par_{dist} = \text{fst } \{q \in p.neigh, q.d + 1 = p.d\}$
Algorithm for the root(`p.root = true`)
`Root` Action: if $p.d \neq 0$ then
$p.d$ is set to 0
Algorithm for any non-root node(`p.root = false`)
CD Action: if $p.d \neq Dist_p$ then
$p.d$ is set to $Dist_p$
CP Action: if $p.d = Dist_p$ and $p.par.d + 1 \neq p.d$ then
$p.par$ is set to Par_{dist}

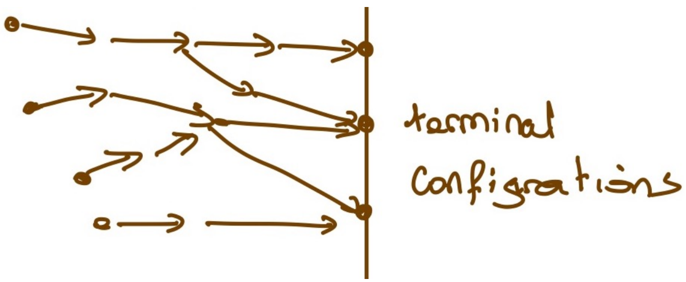
A = { Ap }

Assume = rooted, bidir network
Daemon = unfair
SPEC = the algorithm terminates
terminal config. contains a BFS spanning tree

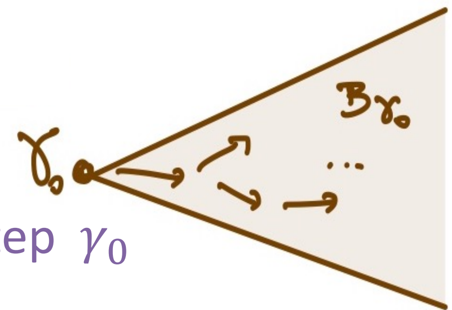
LC

Tools for Convergence

Convergence for this algorithm: every execution is finite



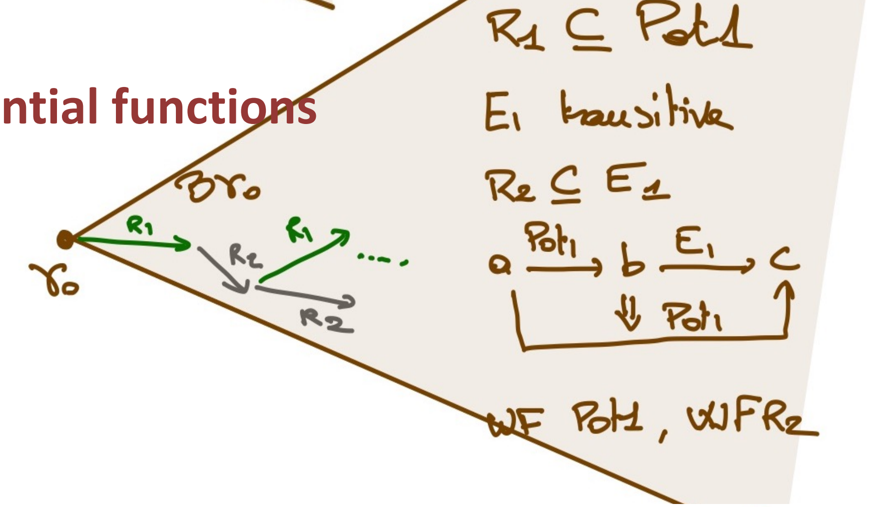
No restriction on the daemon
→ We can directly deal with relation Step
→ Prove that Step is well-founded := WF Step



Fixing an initial configuration γ_0

WF Step := forall γ_0, Acc Step γ_0

Dealing with union of relations and potential functions



Lemma WF_union: forall γ_0,
A sufficient condition for 2 relations,
R1 and R2 being well-founded

$\Rightarrow$ Acc $(R_1 \cup R_2)$ γ_0

Sketch of Proof (Hidden slide)

a . $R_1 \subseteq Pot_1$ e . Trans E_1

b . WF Pot_1 f . WF R_2

c . Stable $a \xrightarrow{S_1} b \rightarrow b \xrightarrow{E_1} c \rightarrow a \xrightarrow{Pot_1} c$

d . $R_2 \subseteq E_1$

¿ WF $(R_1 \cup R_2)$?

① $a \; R_1 \cup R_2 \; b \subseteq (a,a) <_{lex} (b,b)$

where $(a,b) <_{lex} (c,d) \equiv a \xrightarrow{Pot_1} c \vee a \xrightarrow{E_1} c \wedge b \xrightarrow{R_2} d$

uses a and d

② Prove $(a,a) <_{lex} (b,b)$ is WF

by proving that $<_{lex}$ is WF

in general.

uses b c e f

Proof Global Schema

- Step_par = d unchanged
  par can change

  $\Longrightarrow$ WF Step_par

$\uplus$

- Step_d = d values update but the root
  par can change

  $\Longrightarrow$ WF (Step_d U Step_par)

  Uses Lemma WF_union
  Proof Obligation: provide
  - B_γ_0
  - A potential function and WF order for Step_d
  - Potential should not depend on par values

$\uplus$

- Step_root = the root d value changes
  other variable can change

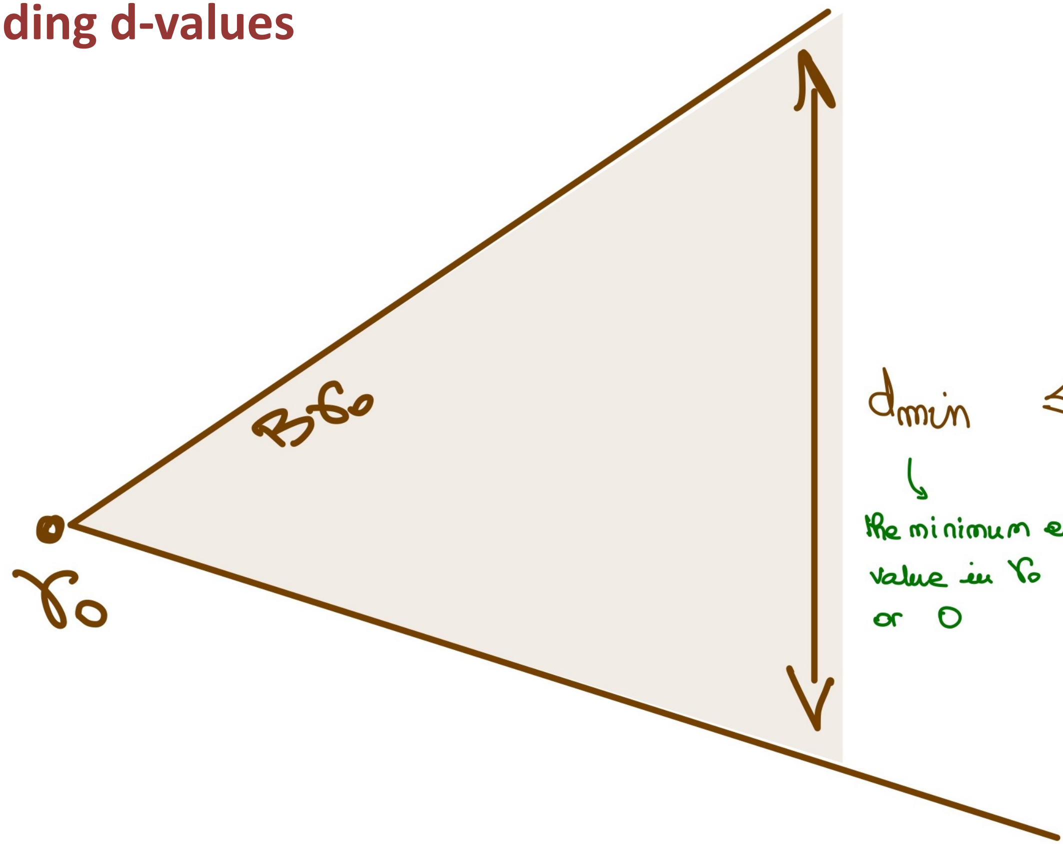
  $\Longrightarrow$ WF (Step_root U
  (Step_d U Step_par))

  Apply Lemma WF_union
  WF Step_root

Potential Function

Proof Obligation: provide
- B_γ_0
- A potential function and WF order for Step_d
- Potential should not depend on par values

Bounding d-values



$d_{min} \le d \le d_{max}$ at p

d_{min}: the minimum existing value in γ_0 or 0

d_{max}: $\max \{ \gamma_0.p.d,$
$1 + \min \{ d_{max}$ at q,
$q \in Neigh \wedge$
$\|p, x\| = 1 + \|q, x\| \} \}$

γ_0 and B_γ_0 are now fixed!

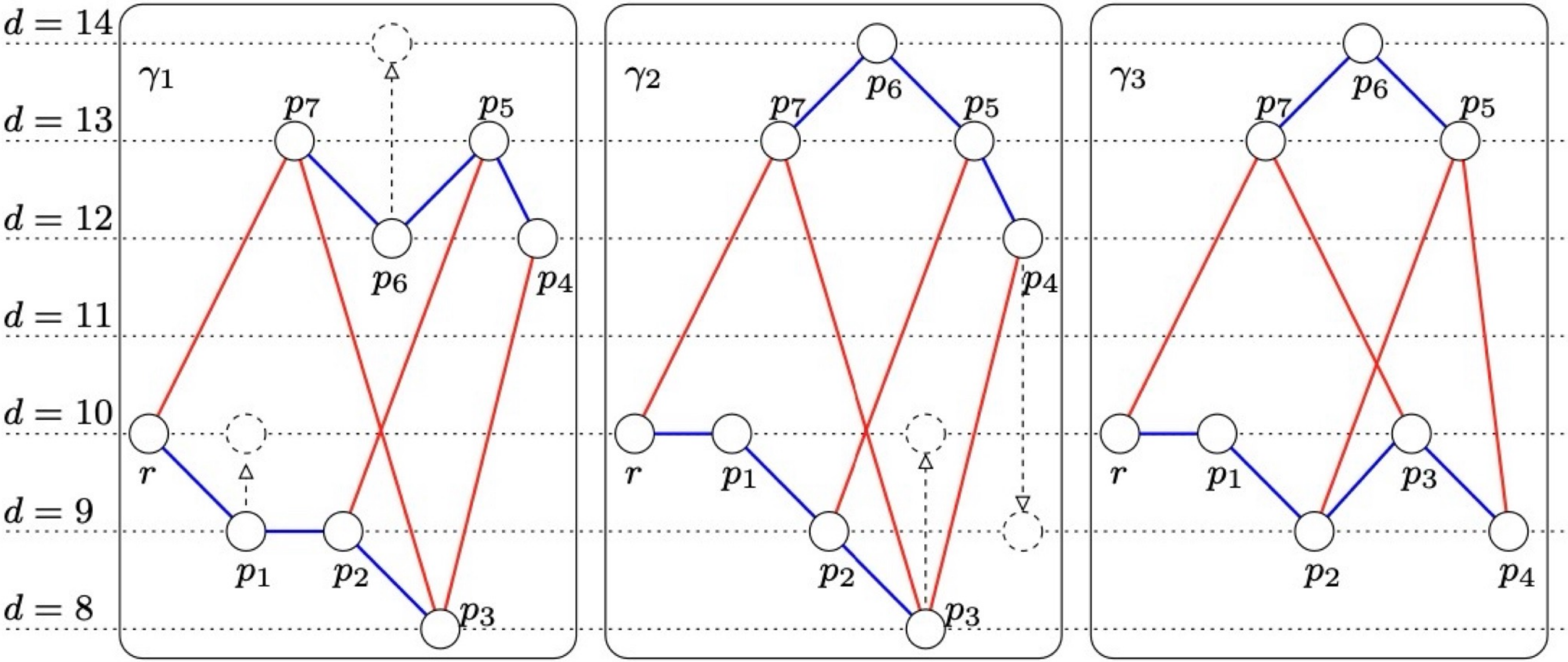
Potential Function

Proof Obligation: provide

- B_γ_0
- A potential function and WF order for Step_d
- Potential should not depend on par values

Smooth edge (p, q) = d values at p and q differ by at most 1

No-Smooth edge (p, q) = …………………………… >= 2



Potential Function

Proof Obligation: provide

- B_γ_0
- A potential function and WF order for Step_d
- Potential should not depend on par values

Smooth Steps

i.e. when only smooth edges change in a Step_d

$\gamma \xrightarrow{\text{Step d}} \gamma'$

Σd increase and d are bounded

Potential function :

$(\text{---------------} , \Sigma d)$

with lexicographic order

Potential Function

Proof Obligation: provide

- B_γ_0
- A potential function and WF order for Step_d
- Potential should not depend on par values

No-Smooth Steps when at least one non-smooth edge changes during a Step_d

- Rank of an edge $(p, q) = \min(p.d,\ q.d)$
- $E_k = \{$ non-smooth edges of rank $k \}$
- $k^* =$ the smallest k for which a non-smooth edge changes

k is bounded by B_{γ_0}

- Potential function $(\underbrace{\ldots E_k \ldots}_{= \ = \ = \ =} \; E_{k^*} \ldots ,\ \Sigma d)$

by def of k^*

$\gamma \xrightarrow{\text{Step}_d} \gamma'$: comparison of E_k by $\subseteq$

+ Lexicographic Order $(\subseteq, \subseteq \ldots, \subsetneq, >)$

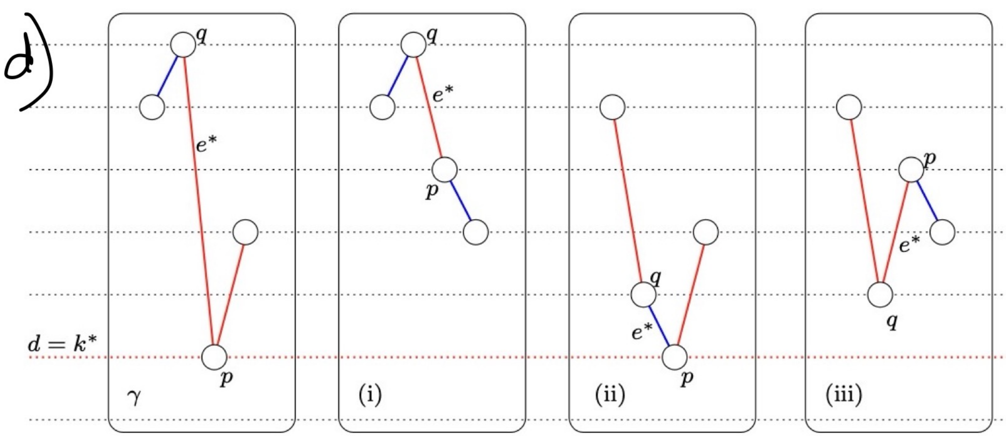


Fig. 2. Possible evolutions of a non-smooth edge $e^* = (p, q)$ with minimal rank k^*: only p executes, (ii) only q executes, (iii) p and q executes.

Comments and Lessons

PADEC Legacy

Faithful to Self-Stabilization
Generic models and tools
Proof simplified using specific proof patterns

A new machine-checked proof for an (old) existing algorithm

Constructiveness
New proof
Better understanding of convergence

Proof assistant is essential

To organize and simplify the proof
To deal with combinatorial explosion of cases
The proof is correct ☺

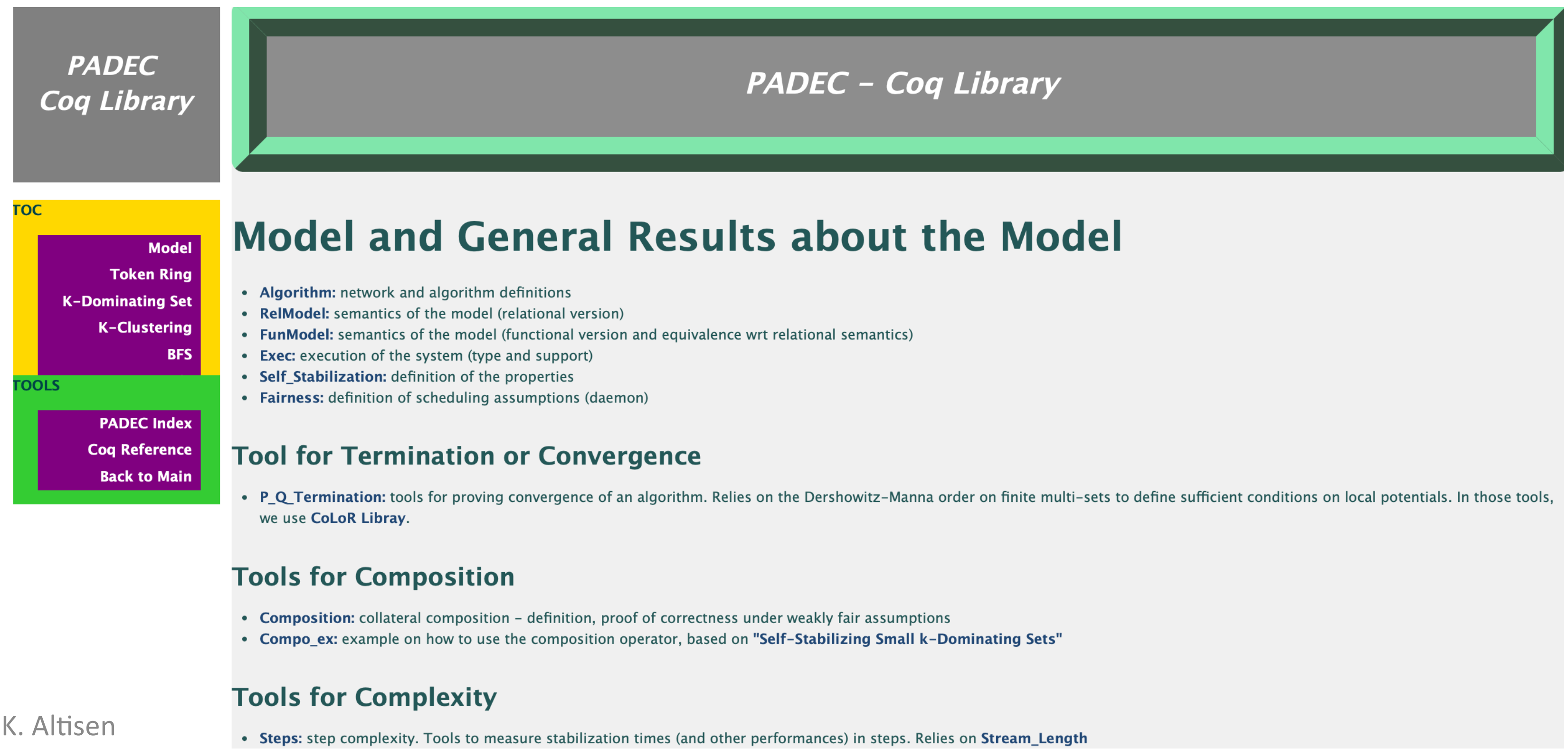
Future work?

use the potential function to obtain complexity in steps

PADEC

http://www-verimag.imag.fr/PADEC-1063.html

#loc = 96k (spec); 33k (proof); 7k (comments)



PADEC Coq Library

TOC

Model
Token Ring
K-Dominating Set
K-Clustering
BFS

TOOLS

PADEC Index
Coq Reference
Back to Main

PADEC - Coq Library

Model and General Results about the Model

- **Algorithm:** network and algorithm definitions
- **RelModel:** semantics of the model (relational version)
- **FunModel:** semantics of the model (functional version and equivalence wrt relational semantics)
- **Exec:** execution of the system (type and support)
- **Self_Stabilization:** definition of the properties
- **Fairness:** definition of scheduling assumptions (daemon)

Tool for Termination or Convergence

- **P_Q_Termination:** tools for proving convergence of an algorithm. Relies on the Dershowitz-Manna order on finite multi-sets to define sufficient conditions on local potentials. In those tools, we use **CoLoR Libray**.

Tools for Composition

- **Composition:** collateral composition - definition, proof of correctness under weakly fair assumptions
- **Compo_ex:** example on how to use the composition operator, based on **"Self-Stabilizing Small k-Dominating Sets"**

Tools for Complexity

- **Steps:** step complexity. Tools to measure stabilization times (and other performances) in steps. Relies on **Stream_Length**

PhD Position, University of Geneva

The University of Geneva – Computer Science Department offers an opportunity to work as a PhD student on **formal methods and distributed algorithms**, under the supervision of Pr. Karine Altisen. The subject will be focused on impossibility results of distributed algorithms, their formalization and formal proofs, using a proof assistant such as Coq/Rocq.

This position involves serving as a teaching assistant for one course per semester (~2h/week of exercises) and goes over 4 years.
A Master's degree (either already obtained or to be completed within the coming months) in Computer Science is required. A strong affinity for discrete mathematics and algorithms is a plus.

Please submit your application to Karine.Altisen@unige.ch, including a CV and the contact of at least one person who may be contacted to support the application.