PADEC
Interactive Proof for Self-Stabilizing Algorithms

Karine Altisen, Pierre Corbineau, Stéphane Devismes

Fontainebleau, le 08/11/2021

How to Gain Confidence into Distributed Algorithms?

Why? Complex statements:
Algorithms, Topologies, Scheduling assumptions...

Pen&paper Proof (usual practice)

Proof = artifact to *convince of the validity* of an assertion

From **[Lamport, How to Write a 21st Century Proof, 2012]**

"Proofs are still written in prose pretty much the way they were in the 17th century. [...]"

"Proofs are unnecessarily hard to understand, and they encourage sloppiness that leads to errors."

How to Gain Confidence into Distributed Algorithms?

Pen&paper Proof (usual practice) → prone to error?

Test, Simulation → few pattern cases

Verification, e.g. Model-Checking → scaling

Machine-checked Proof (proof assistant)

→ heavy development
→ correctness, few convergence
→ very few quantitative properties
→ no complexity

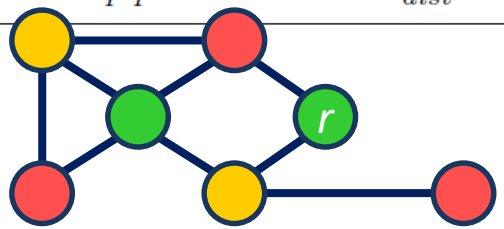
➔ **PADEC**

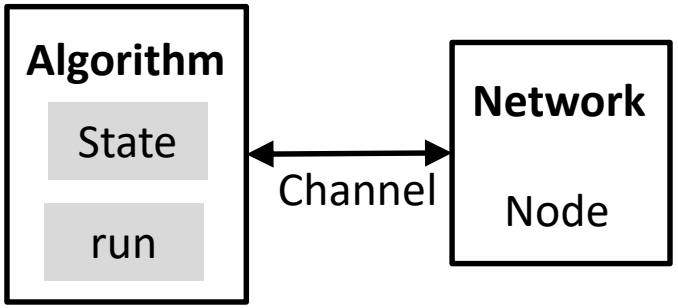
A Coq Framework to Prove *Self-stabilizing* Algorithms in the *Atomic State Model (ASM)*

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., weakly fair*
- **Network**, *e.g. rooted, bidir, connected*

Express **Specification**:
- **Self-stabilizing** w.r.t. a **problem** *e.g., BFS spanning tree*
- **Complexity**, *e.g. convergence requires at most Diameter* ***Rounds***

Prove it!!

Definitions

Examples

PADEC – Big Picture

Common proof patterns & results

Case studies

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

Computational Model
ASM Semantics

KDomSet, KClustering

Relational <->
Functional

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

Unfair, weakly fair, synchronous

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

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

BFS spanning tree
Token circulation
Clustering

Induction Schema

BFS spanning tree (rounds)

Dijkstra Token Ring (steps)

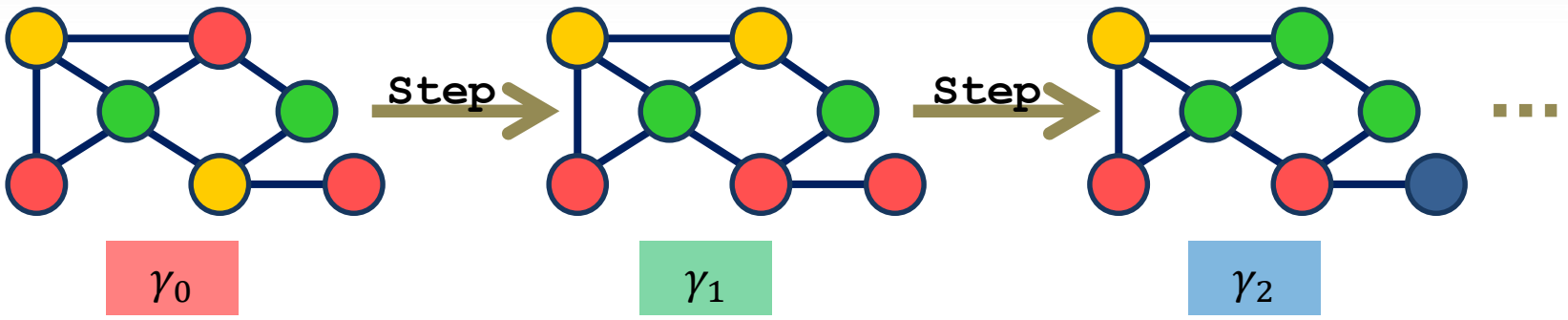
Composition
- Hierarchical Collateral

Proof of
- Specification
- Complexity

BFS + KClustering

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

Computational Model – ASM Semantics



Configuration γ_i : `Env` (state of all nodes `Env := Node -> State`)

Atomic step - read local & neighbor variables → enabled?
- daemon selection
- node computation → update local variables

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

Execution `Exec := Stream Env`
such that (*predicate* `is_exec: Exec -> Prop`)
- Each two consecutive configurations are linked by Step
- if the stream is finite, the last configuration is *terminal*

Relational semantics
<->
Functional semantics

Assumptions about Daemons & Networks

Networks

- Basic properties (bidirectional, connected, rooted)
- Topologies (ring, tree)
- Measures (distance, diameter)

Daemons – model the asynchronism in the ASM model

In PADEC: a *predicate* over executions `Exec -> Prop`

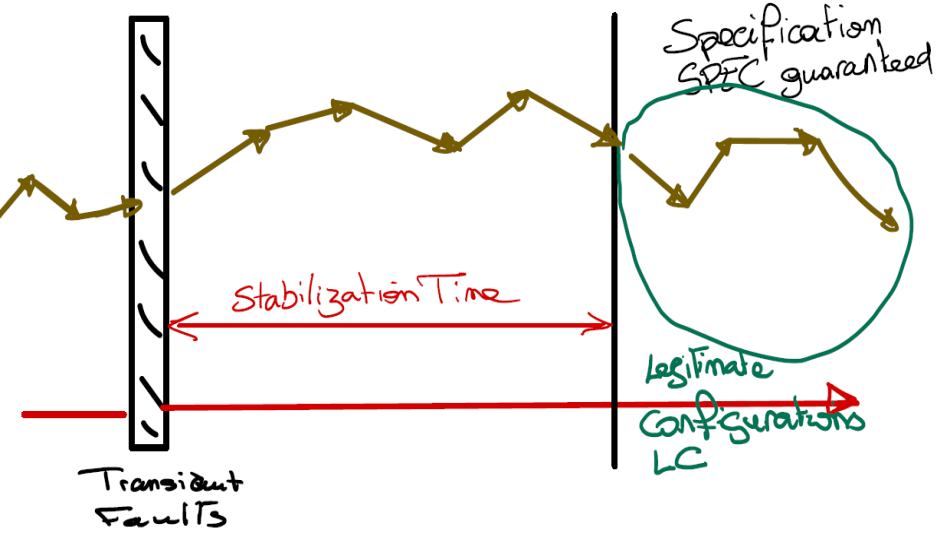
Classical daemons available in PADEC:

unfair, weakly fair, synchronous…

```
unfair e := True (* no constraint *)

weakly_fair e := (* a node which is enabled is
eventually activated or neutralized, and this forever *)
∀ p, Always (fun e => EN p e -> Eventually (AN p) e) e
```

Specification – Self-Stabilization



Defined w.r.t. a problem specification

```
SPEC: Exec -> Prop

self_stab SPEC :=
∃LC: Env -> Prop,
∀e,
```

Closure: if `e` starts in **LC** *then* **Always** `e` remains in **LC**

Convergence: **Eventually** `e` reaches **LC**

Specification: if `e` starts in **LC** then **SPEC** `e`

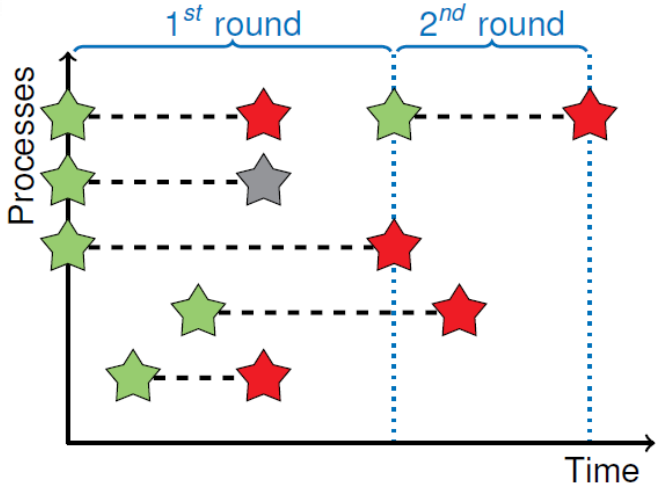
Tools for *Convergence :*
- Lexicographic ordering,
- Well-foundedness,
- Potential & Multiset ordering

Specification – Problem - Complexity

Problems

- *BFS spanning tree*
- *Token circulation*
- *Clustering*

Expressed in `SPEC: Exec -> Prop`



Complexity measures

- **Steps** (number of atomic steps in executions) *Dijkstra Token Ring (steps)*
- **Rounds** *BFS spanning tree (rounds)*

Induction Schema – *e.g. (simplified):*

`P(n): Exec -> Prop` `e: Exec`

If $\forall$ `e`, $\forall$ `n` $\leq$ `B`, `P(n) e -> e` reaches `P(n+1)` in at most one Step/Round
If `P(0) e` holds
Then `e` reaches `P(B)` in at most `B` Steps/Rounds

Hierarchical Collateral Composition

A1;A2

A1 assumes **H1**
is self-stabilizing w.r.t. **SPEC1** and terminates (silent)

A2 shares variables with **A1** but cannot overwrite them
assumes **SPEC1**
is self-stabilizing w.r.t. **SPEC2**

weakly fair daemon (so that **A1** can converge)

Proof of specification: A1;A2 is self-stabilizing, w.r.t. SPEC2 assuming **H1**
(convergence is quite tricky)

Proof of complexity: round complexity is additive in this case
(WIP)

Comments and Lessons

PADEC: a Coq Framework to prove Self-Stabilizing Algorithms

General Model: (not dedicated to a particular case)

Atomic State Model, Daemons, …

→ Close to designer

Reasoning on formal proof: as close as possible of the pen&paper proof

→ Get rid of generality using simplifying tools!

Generic powerful tools: counting, slices, graph properties…

Formal proofs: strengthen assumptions; develop new proofs and sometimes bring new results!

PADEC

http://www-verimag.imag.fr/~altisen/PADEC/

#loc = 14k (spec); 44k (proof); 8k (comments)