PADEC
Interactive Proof for Self-Stabilizing Algorithms
Karine Altisen, Pierre Corbineau, Stéphane Devismes
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)*

K. Altisen, P. Corbineau, S. Devismes
PADEC – Short How To

Algorithm 1 Algorithm BFS, code for each node $p$.

**Constant Local Input:** $p.neigh \subseteq \text{Node}; p.root \in \{t, f\}$

**Local Variables:** $p.d \in \mathbb{N}; p.par \in \text{Node}$

**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 ($p.root = \text{true}$)**

- **Root Action:** if $p.d \neq 0$ then $p.d$ is set to 0

**Algorithm for any non-root node ($p.root = \text{false}$)**

- **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!!

K. Altisen, P. Corbineau, S. Devismes
PADEC – Big Picture

Computational Model
ASM Semantics

Specifcation
- Self-Stabilization
- Problem
- Complexity: Steps, Rounds

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

Composition
- Hierarchical Collateral

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

Assumptions
- Daemons
- Networks

Unfair, weakly fair, synchronous
Connected, ring, tree
Identified, (semi-)anonymous
Measures (distance, diameter)

KDomSet, KClustering
Relational <-> Functional

Proof of
- Specification
- Complexity
BFS + KClustering

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

Case studies
Common proof patterns & results

Examples

Definitions
Computational Model – ASM Semantics

Configuration $\gamma_i$: $\text{Env}$ (state of all nodes $\text{Env} := \text{Node} \rightarrow \text{State}$)

Atomic step
- read local & neighbor variables $\rightarrow$ enabled?
- daemon selection
- node computation $\rightarrow$ update local variables

Relation $\text{Step} := \text{Env} \rightarrow \text{Env} \rightarrow \text{Prop}$

Execution $\text{Exec} := \text{Stream Env}$ such that ($\text{predicate is}_\text{exec}: \text{Exec} \rightarrow \text{Prop}$)
- Each two consecutive configurations are linked by $\text{Step}$
- if the stream is finite, the last configuration is $\text{terminal}$

K. Altisen, P. Corbineau, S. Devismes
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 \( \text{Exec} \rightarrow \text{Prop} \)
Classical daemons available in PADEC:
  - unfair, weakly fair, synchronous...

\[
\text{unfair} \ e := \text{True} \quad (* \text{no constraint} *)
\]

\[
\text{weakly\_fair} \ e := \quad (* \text{a node which is enabled is eventually activated or neutralized, and this forever} *)
\ \forall p, \text{Always} \ (\text{fun} \ e \rightarrow \text{EN} \ p \ e \rightarrow \text{Eventually} \ (\text{AN} \ p) \ e) \ e
\]
Specification – Self-Stabilization

** Defined w.r.t. a problem specification **

$$\text{SPEC: } \text{Exec} \rightarrow \text{Prop}$$

$$\text{self\_stab } \text{SPEC} := \exists \text{LC: } \text{Env} \rightarrow \text{Prop}, \forall e,$$

**Closure:** if \( e \) starts in \( \text{LC} \) then

**Always** \( e \) remains in \( \text{LC} \)

**Convergence:** Eventually \( e \) reaches \( \text{LC} \)

**Specification:** if \( e \) starts in \( \text{LC} \) then

\( \text{SPEC} \ e \)
Specification – Problem - Complexity

Problems
- BFS spanning tree
- Token circulation
- Clustering

Expressed in SPEC: Exec \rightarrow Prop

Complexity measures
- **Steps** (number of atomic steps in executions)
- **Rounds** BFS spanning tree (rounds)

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

\[ P(n) : \text{Exec} \rightarrow \text{Prop} \quad \text{e: Exec} \]

If \( \forall e, \forall n \leq B, P(n) \text{ e } \rightarrow e \text{ reaches } P(n+1) \text{ in at most one Step/Round} \)
If \( P(0) \text{ e holds} \)
Then \( e \text{ reaches } P(B) \text{ in at most } B \text{ Steps/Rounds} \)
Hierarchical Collateral Composition

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)