Composition certifiée d'algorithmes autostabilisants silencieux

Karine Altisen, Pierre Corbineau

VERIMAG UMR 5104, Université Grenoble Alpes, France

19èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (AlgoTel)
Quiberon, 30 Mai-2 Juin 2017

This work was funded by AGIR PADEC & ANR ESTATE

Composition for distributed algorithms

Composition = technique for building complex algorithms

Example Composition $\mathcal{A}_1 ; \mathcal{A}_2$ of sequential algorithms

- $\mathcal{A}_1$ takes an input and computes an output
- Once $\mathcal{A}_1$ has terminated, $\mathcal{A}_2$ takes the output from $\mathcal{A}_1$ and computes its own output

What if $\mathcal{A}_1$ and $\mathcal{A}_2$ are distributed self-stabilising algorithms ?

- We cannot know if $\mathcal{A}_1$ has globally finished computing.
- We have to start executing $\mathcal{A}_2$ anyway.

Questions to address

- How do we compose $\mathcal{A}_1$ transitions with $\mathcal{A}_2$ transitions ?
- Can we formally guarantee that specifications for both $\mathcal{A}_1$ and $\mathcal{A}_2$ will be satisfied by $\mathcal{A}_2 \circ \mathcal{A}_1$?
- Can we formally guarantee convergence of $\mathcal{A}_2 \circ \mathcal{A}_1$?

K. Altisen, P. Corbineau
VERIMAG, Université Grenoble Alpes, France

Goal of the PADEC project

PADEC = *«Preuves d'Algorithmes Distribués En Coq»*
"Proofs of distributed algorithms with Coq"

Goal Establish a library for building formal proofs of distributed algorithms.

Formalism Use Coq and its libraries as a foundation

Methodology

- Define Execution model (Network, Algorithm, Execution, Specification)
- Prove lemmas corresponding to common proof patterns.
- Use case-studies to ensure applicability.

☞ Composition is needed for most non-trivial algorithms

✍ First big case study: k-clustering algorithm [FORTE'2016]

Coq: A generic proof-editing and proof-checking tool

A widely used proof-assistant which features:

- Functional language for definitions
- Interactive proof-editing
- Automated proof-checking

Coq has received the ACM Software System 2013 award.

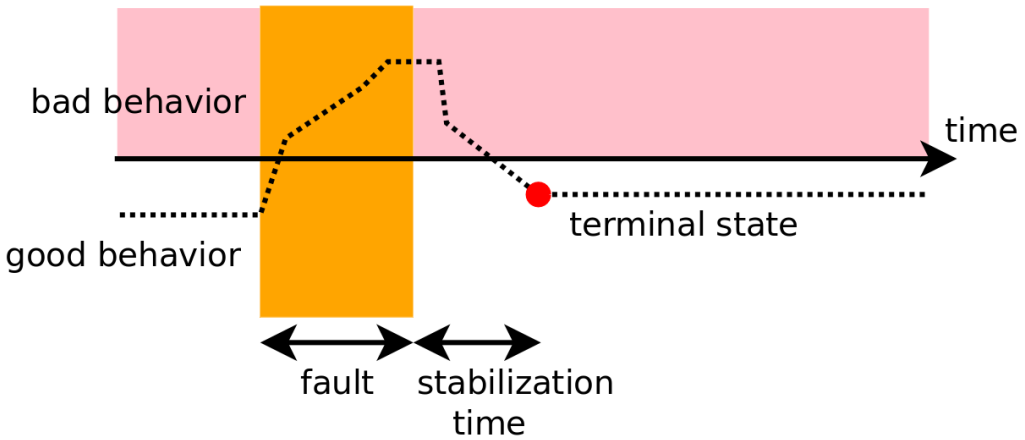
Applications:

- System proofs
  - CompCert certified C compiler [X.Leroy *et al.*]
  - Common Criterion EAL7 certification of industrial systems
- Mathematical proofs
  - Four-colour theorem, odd-order theorem [G. Gonthier *et al.*]

Silent Self-stabilising algorithms

Self-stabilisation: a lightweight solution to fault-tolerance.

- Distributed algorithm on connected graph (network)
- Start from any configuration
- Converge to desired behaviour within finite time
- Need to allow for recovery time

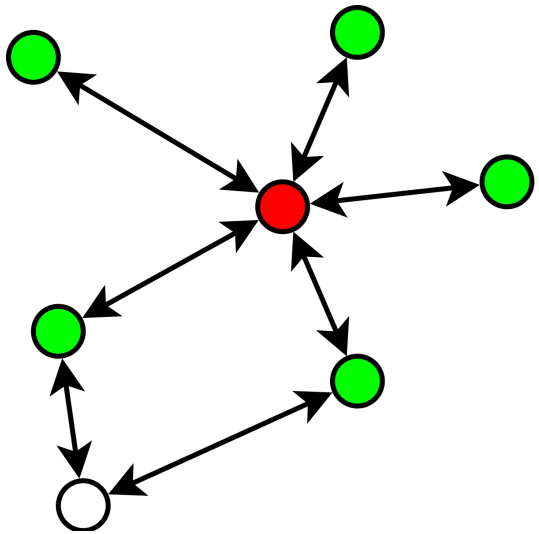


Silent algorithm The algorithm always reaches a (good) terminal configuration in finite time.

Locally-shared Memory Model (1/2)

The algorithm is executed on a distributed system:

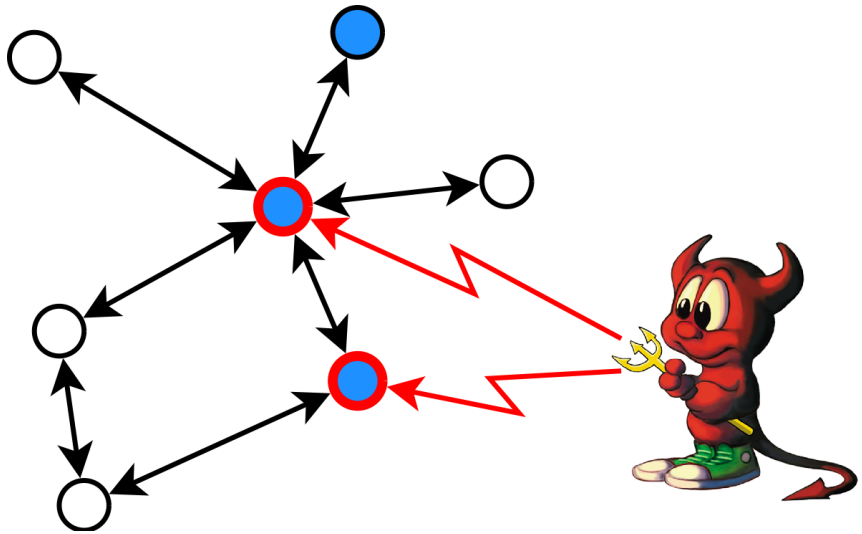
- finite set of interconnected nodes
- each node runs its own code and can modify its own state
- each node can access (read-only) its neighbours' state



Locally-shared Memory Model (2/2)

A node is enabled if it can perform a transition. At each step:

- build the set of all enabled transitions
- if $= \emptyset$ the algorithm terminates
- if $\neq \emptyset$ a scheduler (demon) selects a non-empty subset of enabled transitions to be performed concurrently.



Functional Representation of Algorithms

Operational Representation

Variables:

$n \in \mathbb{N}$

...

Actions:

$\text{Guard}_1 \rightarrow \text{Assign}_1$

$\text{Guard}_2 \rightarrow \text{Assign}_2$

...

Functional Representation

Record State := mkState {
  n:nat;
  ...}

Definition run(s, ℓ) :=
  $\text{Assign}_1(s)$ if $\text{Guard}_1(s, \ell)$
else $\text{Assign}_2(s)$ if $\text{Guard}_2(s, \ell)$
else ...
else $\perp$

Signature: run(state,neighbours' states) $\rightarrow$ (state $|\ \perp$)

Collateral Hierarchical Composition (1/2)

Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two algorithms with transition functions $\text{run}_{\mathcal{A}_1}$ and $\text{run}_{\mathcal{A}_2}$.

- We build $\mathcal{A}_2 \circ \mathcal{A}_1$, the Collateral Hierarchical Composition of $\mathcal{A}_1$ and $\mathcal{A}_2$ [Herman, Devismes]

State a combination of an $\mathcal{A}_1$-state and an $\mathcal{A}_2$-state

☞ The shared information is the output of $\mathcal{A}_1$ and input for $\mathcal{A}_2$

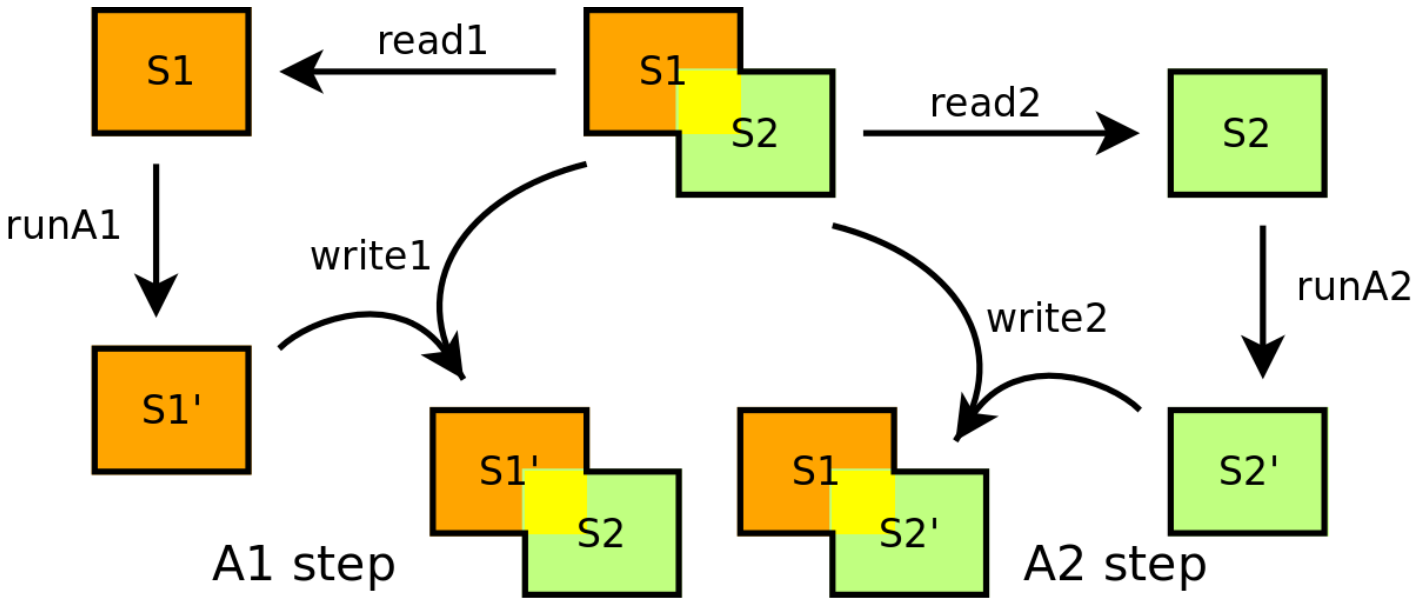
Transitions $\mathcal{A}_1$ has local priority over $\mathcal{A}_2$

1. Execute $\mathcal{A}_1$ if enabled
2. Execute $\mathcal{A}_2$ if enabled and $\mathcal{A}_1$ is disabled

Information flow We assume $\mathcal{A}_2$ transitions cannot modify the $\mathcal{A}_1$ states (only read them).

Collateral Hierarchical Composition (2/2)

We express state modifications using functional constructions



Formal definition of $\mathcal{A}_2 \circ \mathcal{A}_1$

$$\begin{array}{lll} \mathsf{run}_{\mathcal{A}_2 \circ \mathcal{A}_1}(s, \ell) & := & \mathsf{write}_1(s'_1, s) \texttt{ if } \mathsf{run}_{\mathcal{A}_1}(\mathsf{read}_1(s), \mathsf{read}_1 \circ \ell) = s'_1; \\ & \texttt{else} & \mathsf{write}_2(s'_2, s) \texttt{ if } \mathsf{run}_{\mathcal{A}_2}(\mathsf{read}_2(s), \mathsf{read}_2 \circ \ell) = s'_2; \\ & \texttt{else} & \bot \end{array}$$

Partial Correctness

Given a terminal configuration γ for $\mathcal{A}_2 \circ \mathcal{A}_1$:

- In γ, all nodes are disabled for $\mathcal{A}_2 \circ \mathcal{A}_1$, *i.e.* $\text{run}_{\mathcal{A}_2 \circ \mathcal{A}_1}(\_, \_) = \bot$
- Hence all nodes are disabled for both $\mathcal{A}_1$ and $\mathcal{A}_2$
- Thus $\text{read}_1 \circ \gamma$ satisfies the specification for $\mathcal{A}_1$
- Similarly, $\text{read}_2 \circ \gamma$ satisfies the specification for $\mathcal{A}_2$

☞ Thus if $\mathcal{A}_2 \circ \mathcal{A}_1$ converges, its terminal configuration will satisfy specifications for both $\mathcal{A}_1$ and $\mathcal{A}_2$.

✍ Now we must establish the convergence, *i.e.* any $\mathcal{A}_2 \circ \mathcal{A}_1$ computation reaches a terminal configuration after a finite number of steps.

Convergence of collateral hierarchical composition

In general, convergence of $\mathcal{A}_2 \circ \mathcal{A}_1$ doesn't follow from convergence of $\mathcal{A}_1$ and $\mathcal{A}_2$:

- Convergence of $\mathcal{A}_2$ is only assumed from a terminal configuration of $\mathcal{A}_1$.
- If $\mathcal{A}_1$ has not converged yet, $\mathcal{A}_2$ may not converge.
- Unfair scheduling may induce a livelock of $\mathcal{A}_1$ by executing only $\mathcal{A}_2$ forever

Two solutions:

1. Keep the unfair scheduling and assume $\mathcal{A}_2$ converges from any input configuration
2. Prove convergence only for weakly-fair executions.

Convergence under asynchonous (unfair) scheduling

We assume $\mathcal{A}_2$ converges from any input configuration:

- We assume all $\mathcal{A}_1$ steps are decreasing for a relation $\preceq_1$ that is well-founded.
- We assume all $\mathcal{A}_2$ steps are decreasing for a relation $\preceq_2$ that is well-founded, even when starting from a non-terminal $\mathcal{A}_1$ configuration

Lexicographic ordering $\prec_{\mathsf{LEX}}$ for $\mathcal{A}_2 \circ \mathcal{A}_1$ configurations γ,γ'

$\gamma' \prec_{\mathsf{LEX}} \gamma$ holds iff either (1.) $\mathsf{read}_1 \circ \gamma' \prec_1 \mathsf{read}_1 \circ \gamma$ or
(2.) $\mathsf{read}_1 \circ \gamma' = \mathsf{read}_1 \circ \gamma \wedge \mathsf{read}_2 \circ \gamma' \prec_2 \mathsf{read}_2 \circ \gamma$

By construction, all $\mathcal{A}_2 \circ \mathcal{A}_1$ steps are decreasing according to the $\prec_{\mathsf{LEX}}$ relation:

1. Either one node at least performs an $\mathcal{A}_1$ transition
2. Or all activated nodes (at least one), perform $\mathcal{A}_2$ transitions

Since $\prec_{\mathsf{LEX}}$ is well-founded, $\mathcal{A}_2 \circ \mathcal{A}_1$ converges.

Convergence under weakly fair scheduling (1/3)

In general, assuming $\mathcal{A}_2$ converges from any $\mathcal{A}_1$ configuration is too strong a hypothesis.

- $\mathcal{A}_1$ is often used to build a topological structure used to guide the execution of $\mathcal{A}_2$, e.g. spanning tree, ring ...
- Prior termination of $\mathcal{A}_1$ is a requirement for $\mathcal{A}_2$ to converge

We have to assume weekly-fair scheduling to ensure convergence.

Weakly-fair scheduling

An execution is weakly fair iff every node enabled at some point is eventually:

- Activated, *i.e.* allowed to perform a transition, or
- Neutralised, *i.e.* it becomes disabled because of neighbouring transitions

Convergence under weakly fair scheduling (2/3)

Some useful Coq definitions:

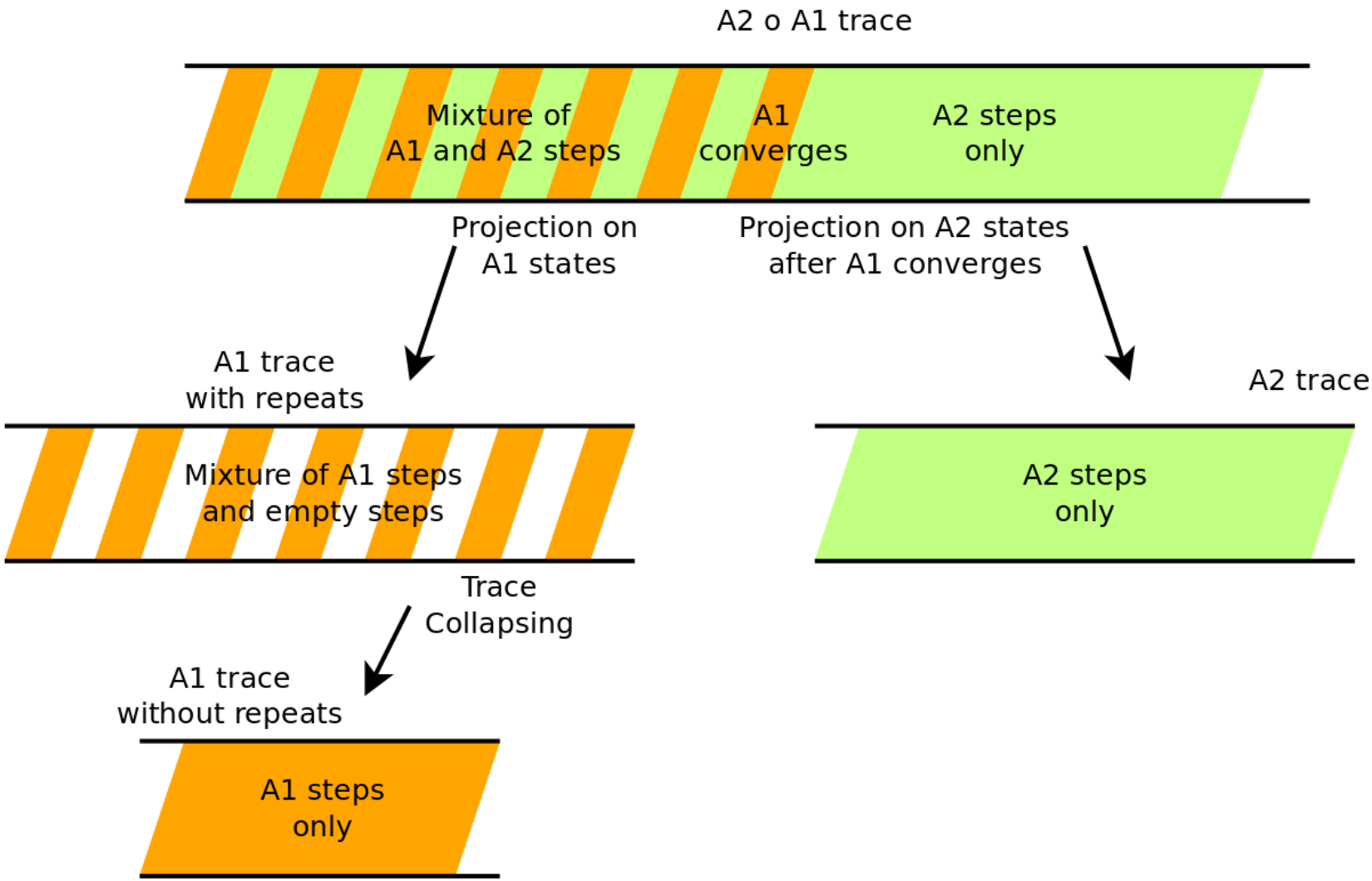
```
CoInductive Stream (A: Type): Type :=
| s_one: A -> Stream A
| s_cons: A -> Stream A -> Stream A.

Inductive Eventually {A: Type}
          (P: Stream A -> Prop) : Stream A -> Prop :=
| eventually_now: forall s, P s -> Eventually P s
| eventually_later: forall a s, Eventually P s ->
                                Eventually P (s_cons a s).

CoInductive Always {A: Type}
          (P: Stream A -> Prop) : Stream A -> Prop :=
| always_one: forall a, P (s_one a) -> Always P (s_one a)
| always_cons: forall a s, P (s_cons a s) ->
                           Always P s -> Always P (s_cons a s).
```

Convergence under weakly fair scheduling (3/3)

Proof sketch:



Conclusion

Composition A commonly used element for building complex self-stabilising algorithms

Certified Composition A key element for proving correctness of complex self-stabilising algorithms in the PADEC library

- Formal definition of Collateral Hierarchical Composition
- Proof of self-stabilisation properties
- Proof of convergence:
  - In the asynchronous case with a strong assumption
  - In the weakly-fair case

DONE complexity properties in rounds

TODO complexity properties in steps

TODO case where $\mathcal{A}_1$,$\mathcal{A}_2$ are non-silent

Thank you !
Any questions ?

K. Altisen, P. Corbineau
VERIMAG, Université Grenoble Alpes, France