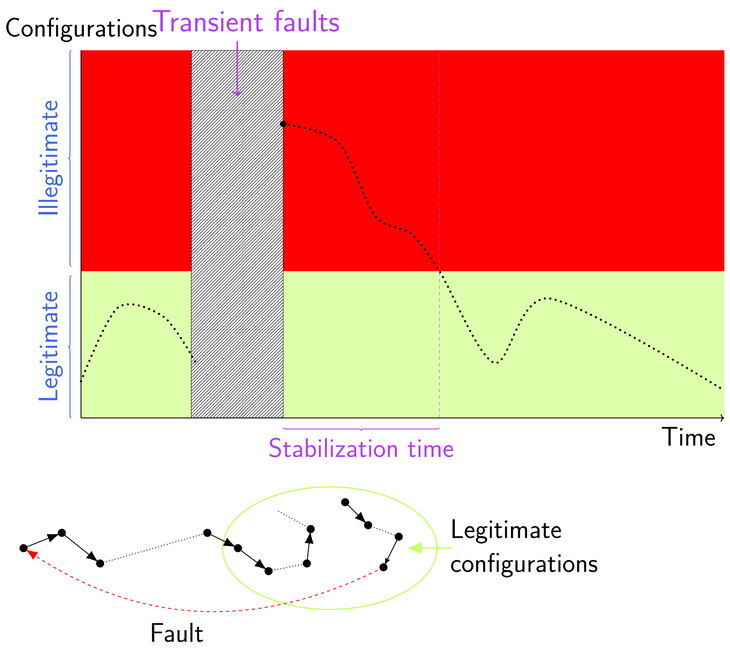
SASA: A SimulAtor of Self-stabilizing Algorithms

Verimag Laboratory

Self-Stabilizing Algorithms

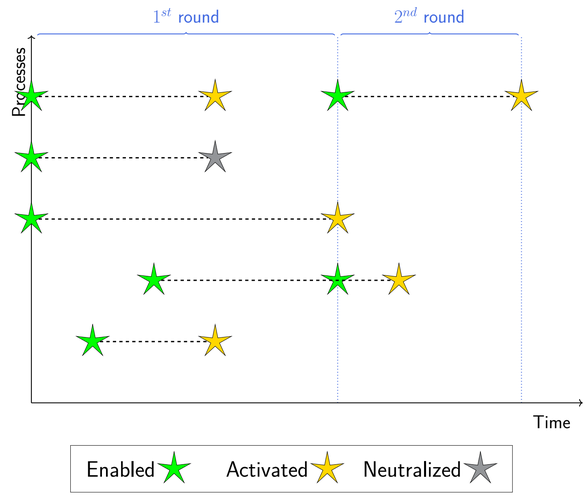


Goal: Study Algorithmic Complexity

- Space complexity: memory requirement in bits
- Time complexity until stabilization
  - in steps, or moves
  - in rounds: execution time of the slowest processes

How?

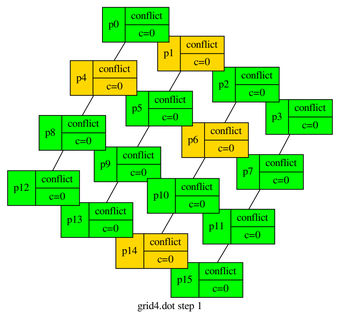
- Paper-and-pencil Proofs
- Proof Assistants (coq)
- Simulations

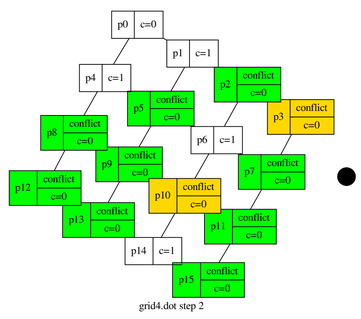


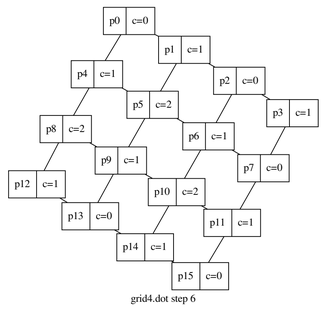
An Example: Coloring Algorithm in the Atomic-State Model (ASM)

- Parameters:
  $p.N$: the set of p's neighbors
  K : an integer such that $K \geq \Delta$
- Local Variable:
  $p.c \in \{0, ..., K\}$ holds the color of parameters
- Macros:
  $Used(p) = \{q.c : q \in p.N\}$
  $Free(p) = \{0, ..., K\} \setminus Used(p)$
  $Conflict(p) = \exists q \in p.N : q.c = p.c$
- Action:
  Color :: Conflict(p) $\hookrightarrow p.c \leftarrow min(Free(p))$

```
open Algo
let k = max_degree ()
let init_state _ = Random.int k
let neigbhors_vals nl = List.map (fun n -> state n) nl
let conflict v nl = List.mem v (neigbhors_vals nl)
let free nl =
  let used = List.sort_uniq compare (neigbhors_vals nl) in
  let rec aux free used i =
    if i > k then free else
      (match used with
       | x::tail -> if x=i then aux free tail (i+1)
                    else aux (i::free) used (i+1)
       | [] -> aux (i::free) used (i+1)
      )
  in
  List.rev (aux [] used 0)
let enable_f e nl = if (conflict e nl) then ["conflict"] else []
let step_f e nl a = List.hd (free nl)
```



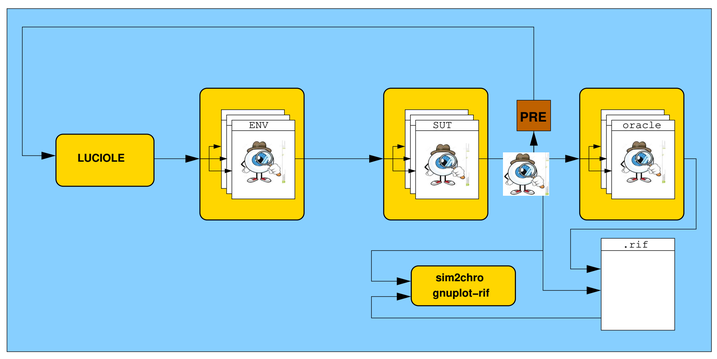




Main Features

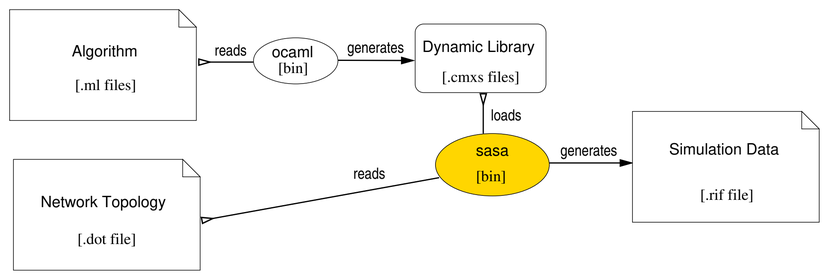
- Batch Simulations
  - Test Algorithms
  - Perform simulation campaigns
    - Study the influence of some parameters
    - Estimate the (average-case) complexity Lower bounds
- Test oracles to formalize expected properties
  - involve the number of steps, moves, or rounds to reach a legitimate configuration
- Daemon can be configured
  - Predefined: synchronous, central, locally central, or distributed
  - Custom daemons: manual or programmed
- Interactive Simulations
  - Debug Algorithms
  - Step by step, round by round, forward or backward
  - While visualizing the network, the enabled, the activated actions
  - New commands can programmed

Reactive Programs Simulation Tools Set

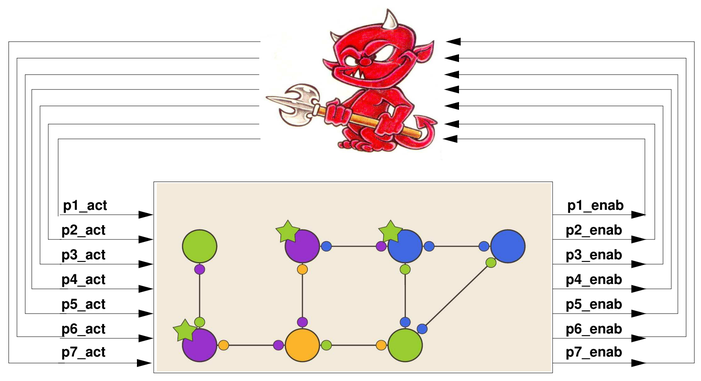


Lustre, Lutin, Lurette, rdbg

The Core Simulator Architecture



ASM Algorithms as Reactive programs



SASA

- An open-source SimulAtor of Self-stabilizing Algorithms
- Written using the Atomic-State Model (the usual model in the Self-Stabilizing community)
- Relies on existing tools as much as possible
  - `dot` for Topologies
  - `ocaml` for programming local algorithms
  - Verimag Tools for oracles, programmable daemons, interactive simulations
- Installable via `docker`, `opam`, or `git`