Soyez Efficace, Rembobinez
Be efficient, Rewind

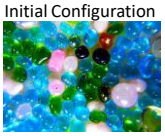
Stéphane Devismes (Verimag, UGA)
Stephane.Devismes@univ-grenoble-alpes.fr
Colette Johnen (LaBRI, Univ. Bordeaux)
johnen@labri.fr

UGA Univ. Grenoble Alpes | Verimag | université de BORDEAUX | LaBRI

1

Distributed Algorithm

Initial Configuration



Goal of the algorithm

Distributed Algorithm

2

Unison : clocks synchronization

Each process u has a clock : c(u)

- the difference of clock values between neighbors is at most 1 (safety)
- Each process increments its clock infinitely often (liveness)

Distributed algorithm using **a bounded** clock –
a bound on network size is known (K)

$\forall$v in the neighborhood of u, we have c(v) $\in$ {c(u), c(u)+1%K}
$\rightarrow$ c(u) := (c(u)+1) %K

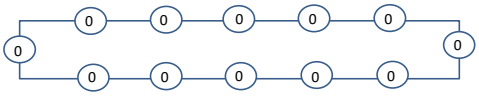
The initial configuration is a safe configuration

K > size of networks

3

Unison : clocks synchronization

$\forall$v in the neighborhood of u, we have c(v) $\in$ {c(u), c(u)+1%K}
$\rightarrow$ c(u) := (c(u)+1) %K



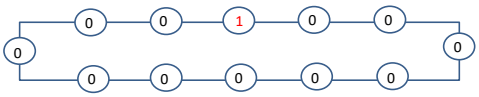
The initial configuration is a safe configuration

K > size of networks

4

Unison : clocks synchronization

$\forall$v in the neighborhood of u, we have c(v) $\in$ {c(u), c(u)+1%K}
$\rightarrow$ c(u) := (c(u)+1) %K



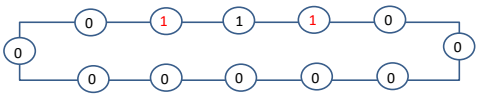
The initial configuration is a safe configuration

K > size of networks

5

Unison : clocks synchronization

$\forall$v in the neighborhood of u, we have c(v) $\in$ {c(u), c(u)+1%K}
$\rightarrow$ c(u) := (c(u)+1) %K



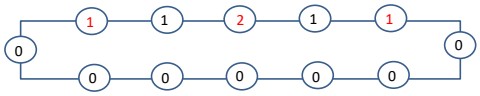
The initial configuration is a safe configuration

K > size of networks

6

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K



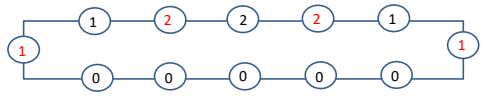
The initial configuration is a safe configuration

K > size of networks

7

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K



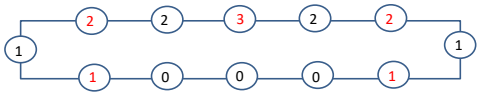
The initial configuration is a safe configuration

K > size of networks

8

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K



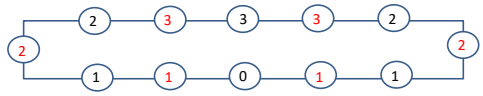
The initial configuration is a safe configuration

K > size of networks

9

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K



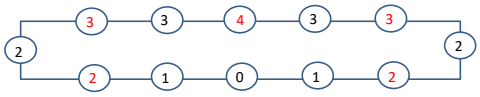
The initial configuration is a safe configuration

K > size of networks

10

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K



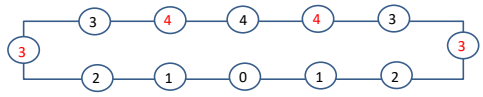
The initial configuration is a safe configuration

K > size of networks

11

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K



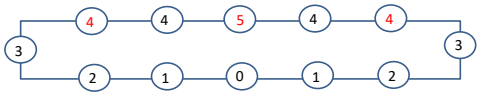
The initial configuration is a safe configuration

K > size of networks

12

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K



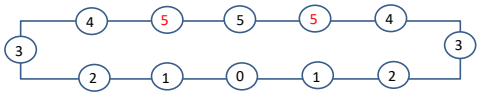
The initial configuration is a safe configuration

K > size of networks

13

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K



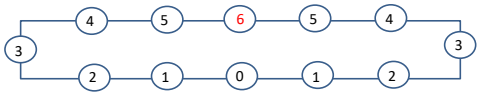
The initial configuration is a safe configuration

K > size of networks

14

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K

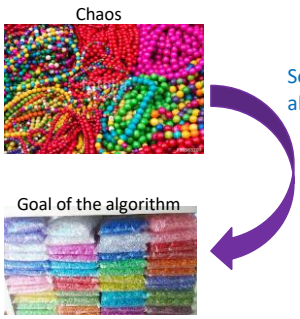


The initial configuration is a safe configuration

K > size of networks

15

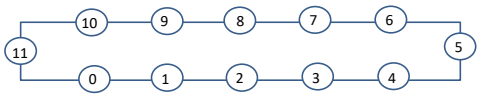
Self-stabilizing algorithm



16

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K



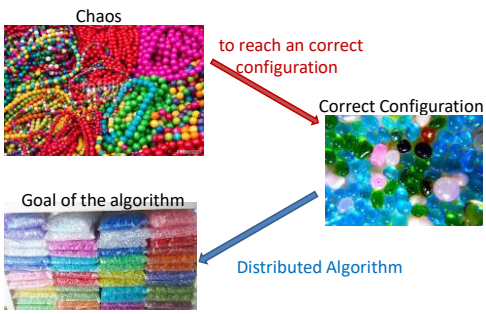
DEADLOCK : the configuration is not a safe configuration

Unison algorithm is not self-stabilizing

K > 12

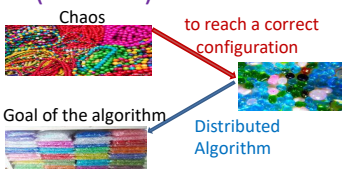
17

To rewind (To reset) To reuse



18

To rewind (To reset) To reuse



- Awerbuch, Patt-Shamir, Varghese: self-stabilization by local checking and global reset (extended abstract). FOCS '91
- Awerbuch, Patt-Shamir, Varghese, Dolev : self-stabilization by local checking and global reset (extended abstract). WDAG'94

19

Self-stabilizing reseting algorithm

Arora, Gouda: Distributed reset. IEEE Trans. Computer. 1994

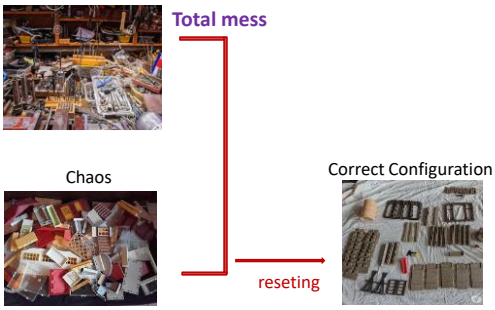
- identified network
- mono initiator
- weakly fair scheduler

SDR algorithm - Self-stabilizing Distributed Reset algorithm

- anonymous network,
- no network knowledge
- multi initiators
- unfair scheduler
- linear number of process moves
- Unbounded timestamps

20

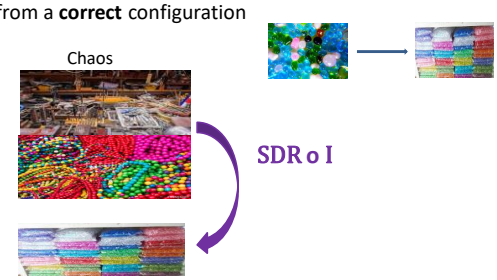
Self stabilizing reseting



21

Chaos → Correct behavior

I : distributed algorithm solving a task (static or dynamic) from a **correct** configuration



22

Correct configuration of I

I : distributed algorithm solving a task (static or dynamic) from a **correct** configuration

Correct configuration of I

P_ICorrect(u) : predicate on u and u's neighbor variables

P_ICorrect(u) is closed along I

∀ u we have (P_ICorrect(u) == true) ≡ the configuration is correct (≠ legitimate)

Pr(u) is closed along I iff Pr(u) stays verified along any execution of I

23

Local reseting in I

I : distributed algorithm solving a task (static or dynamic) from a **correct** configuration

Local reseting in I

reset(u) : macro reseting the value of u' variables

P_reset(u) : predicate on u's variable : it is true iff u's variable are reseted

If u and all u's neighbors verify P_reset then P_ICorrect(u) == true

24

Unison – resetable ?

$\forall$v in the neighborhood of u, we have c(v) $\in$ {c(u), c(u)+1%K}
$\rightarrow$ c(u) := (c(u)+1) %K

P_ICorrect(u) is the safety predicate
$\forall$v in the neighborhood of u, we have
c(v) $\in$ {c(u)-1%K, c(u), c(u)+1%K}

reset(u) : c(u) := 0
P_reset(u) : c(u) == 0
If u and all u's neighbors verify c==0 then
P_ICorrect(u)==true

K > size of networks

25

SDR – Overall presentation

- A process u starts the reseting because
  P_ICorrect(u) != true
- The reseting is propagated (a DAG rooted at u is built)
- The DAG is frozen from the leaves to the initiator of the reseting (u)
- Processes go back to the Initial algorithm/task from the initiator (u) to the leaves of the DAG

26

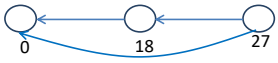
SDR – variables on u

- To store the DAG structure :
  - d(u) : distance to an initiator of the reseting

27

SDR – variables on u

- To store the DAG structure :
  - d(u) : distance to an initiator of the reseting



28

SDR – variables on u

- To store the DAG structure :
  - d(u) : distance to an initiator of the reseting
- The status of the reseting : st(u) : RB, RF, or C
  - a reseting is in progress
    - RB : propagation of the reseting
    - RF : propagation of the ending of broadcast phase
  - no reseting is in progress : C

29

SDR – variables on u

- To store the DAG structure :
  - d(u) : distance to an initiator of the reseting
- The status of the reseting : st(u) : RB, RF, or C
  - a reseting is in progress
    - RB : propagation phase
    - RF : freezing phase
  - no reseting is in progress : C

30

SDR – Self-stabilizing Distributed Reset

R_R : a process u starts the reseting because
P_ICorrect(u) != true or …

R_R: If (st(u)==C) and ¬P_ICorrect(u) →
reset(u); st(u) := RB; d(u) := 0;

A process **performs at most one time R_R rule**
a process can be the root of a single DAG

At most n DAG structure is built during any execution
(n being the network size)

31

SDR – Self-stabilizing Distributed Reset

R_B : the reseting is propagated (a DAG is built)

R_B : If (st(u)==C) and a' u neighbor v verified (st(v)==RB)→
reset(u); st(u) := RB; d(u) := d(v)+1;

A process cannot **join two times the same DAG structure**

A process performs at most n-1 times the rule R_B
(n being the network size)

32

SDR – Self-stabilizing Distributed Reset

R_F: the DAG is frozen from the leaves to the initiator of the reseting

R_F: If (st(u)==RB) and
the reseting propagation is over in the DAG rooted at u
→ reset(u); st(u) := RF;

A process takes the status RB at most n times

A process performs at most n+1 times the rule R_F
(n being the network size)

33

SDR – Self-stabilizing Distributed Reset

R_C : Processes goes back to the Initial algorithm/task

R_C: If (st(u)==RF), u is a root of the DAG and
the reseting is over in the u-DAG →
st(u) := C;

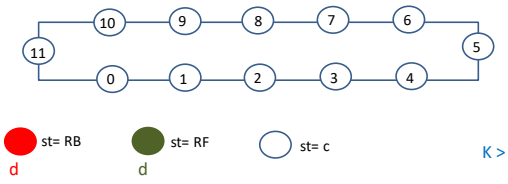
A process takes the status RF at most n times

A process performs at most n times the rule R_C
(n being the network size)

34

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K

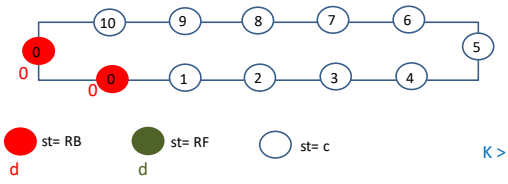


An Unsafe Configuration

35

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K

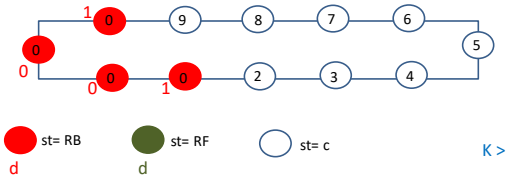


Reseting Propagation phase

36

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K

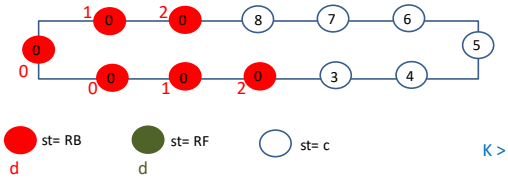


Reseting Propagation phase

37

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K

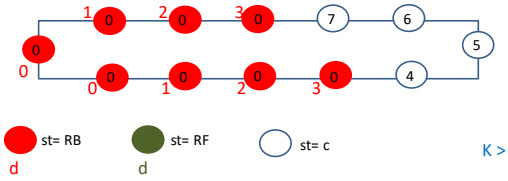


Reseting Propagation phase

38

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K

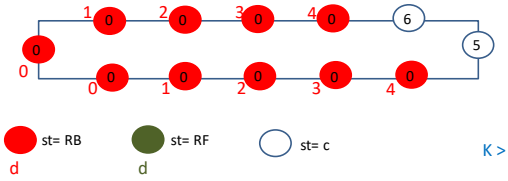


Reseting Propagation phase

39

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K

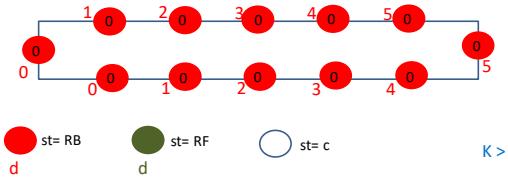


Reseting Propagation phase

40

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K

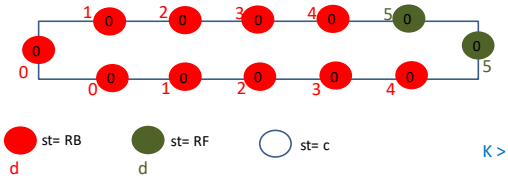


End of Reseting Propagation phase

41

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K

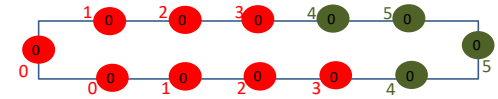


Freezing phase

42

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K



st= RB st= RF st= c K > 12

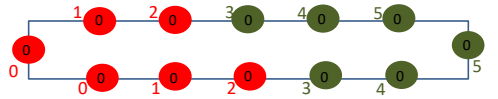
d d

Freezing phase

43

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K



st= RB st= RF st= c K > 12

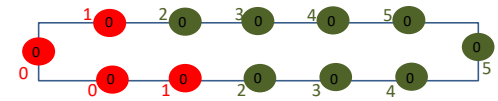
d d

Freezing phase

44

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K



st= RB st= RF st= c K > 12

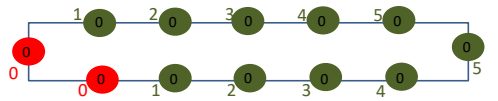
d d

Freezing phase

45

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K



st= RB st= RF st= c K > 12

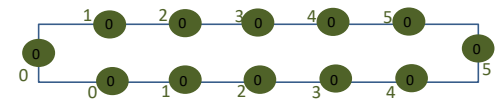
d d

Freezing phase

46

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K



st= RB st= RF st= c K > 12

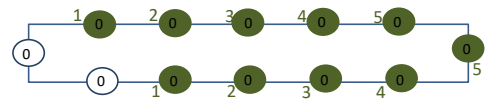
d d

End of Freezing phase

47

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K



st= RB st= RF st= c K > 12

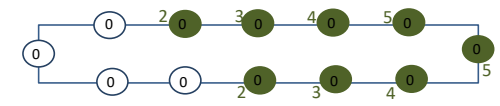
d d

Back to the Initial Algorithm

48

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K

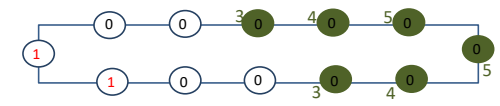


st= RB d st= RF d st= c K > 12

Back to the Initial Algorithm

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K

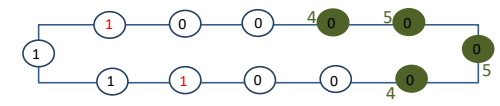


st= RB d st= RF d st= c K > 12

Back to the Initial Algorithm

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K

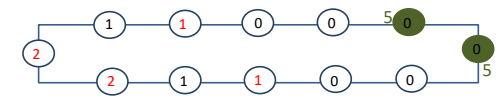


st= RB d st= RF d st= c K > 12

Back to the Initial Algorithm

Unison : clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K

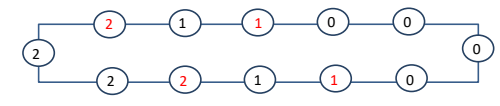


st= RB d st= RF d st= c K > 12

Back to the Initial Algorithm

Unison : clocks synchronization

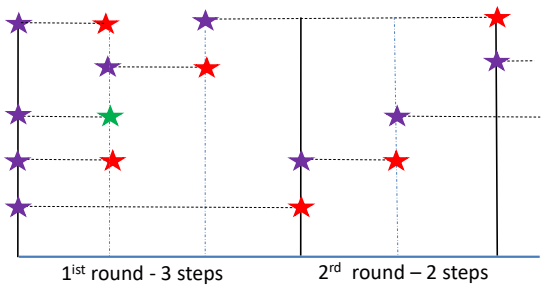
∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K}
→ c(u) := (c(u)+1) %K



st= RB d st= RF d st= c K > 12

A Safe Configuration

Step/Round - illustration



Self-stabilizing Unison

$\mathcal{D}$: network Diameter n : network size

algorithm	memory	stops	rounds
[1]	O(n^2)	-	O($\mathcal{D}n$) [Boulinier PhD]
[2]	O(n)	O($\mathcal{D}n^3$) [3]	O(n)
SDR o I	unbounded	O($\mathcal{D}n^2$)	$3n$

[1] Couvreur, Francez, Gouda: Asynchonous unison (extended abstract), ICDCS'92
[2] Boulinier, Petit, Villain: When graph theory helps self-stabilization, PODC'04
[3] Devismes, Petit: On efficiency of Unison, TADDS'12

THANK YOU

Technical report on HAL:
Self Stabilizing Distributed Cooperative Reset.

ICDCS'19