Relationships between communication models based on registers for fault-tolerant distributed computing on networks

Colette Johnen, Lisa Higham
Univ. Bordeaux, CNRS, LaBRI, UMR 5800
Univ. Calgary, Canada

Semantics of a register

[Lamport 86]

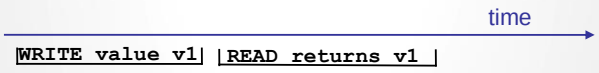
Single-Writer Register (SR)- def

- A register is a memory cell on which two types of operations are possible **READ** and **WRITE**
- On a Single-Writer register, only one process can do the **WRITE** operation
- On a register R, **READ** and **WRITE** operation are not atomic, they take some time

A **READ** operation on a register R may overlap several **WRITE** operations on R

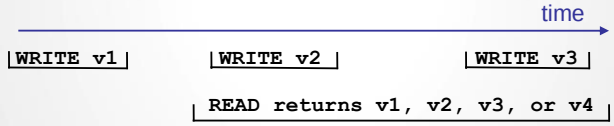
Single-Writer register - semantic

- On R, a **READ** operation that does not overlap any **WRITE** operation returns the most recent preceding written value (v1) in R



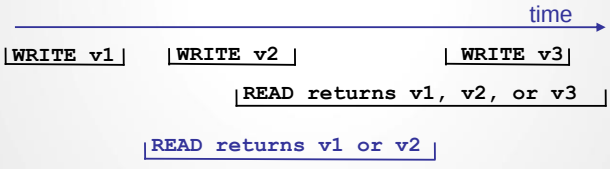
Safe register [Lamport 86]

- On R, a **READ** operation that does overlap a **WRITE** operation may return any value



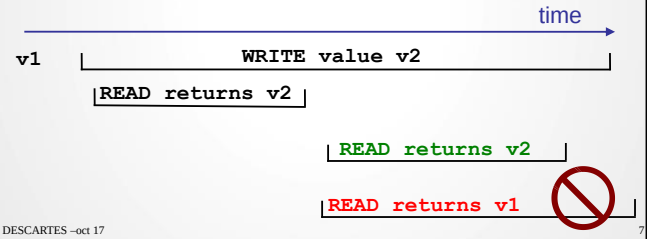
Regular and Atomic register [Lamport 86]

- On R, a **READ** operation that does overlap a **WRITE** operation returns the most recent preceding written value or any value written during overlapping **WRITE** operations



Atomic register [Lamport 86]

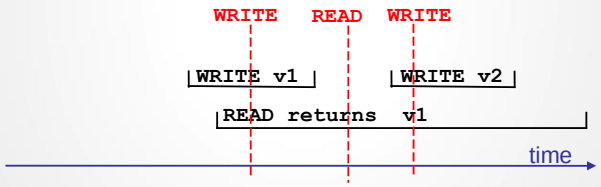
- On R, if a **READ** operation returns the value written during the overlapping **WRITE** operation then any subsequence **READ** cannot return the most recent preceding written value (v1)



Property of atomic registers

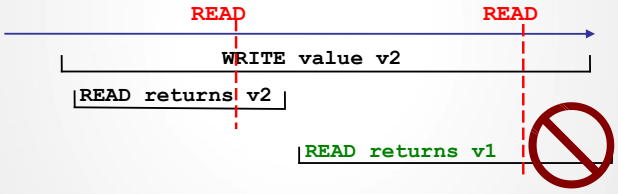
- A sequence of operations on an atomic register is linearizable [Herlihy, Wing 90]

  Each operation appears to happen instantaneously at some point during its execution



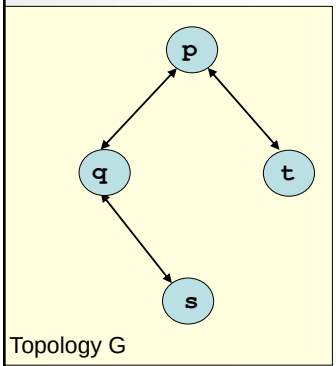
Regular register

- A sequence of operations on an regular register may not linearizable



Communication Model on networks

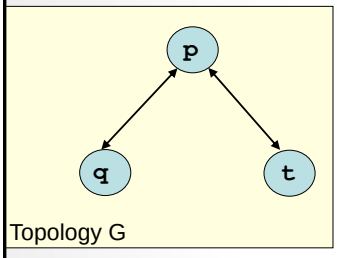
Distributed computing on networks



A processor can only communicate with its neighbors

Ex: *p* can only communicate with *q* and *t*

Communication modelS based on Single-Writer registers



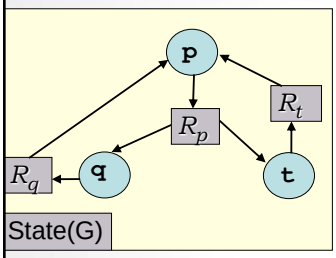
- a single register per process : multi-reader register

  [MN98], [AS99], [H00], [NA02]

- a register is associated to a link : single-reader register

  [DIM93], [Dolev 02]

a single register per process : state network model

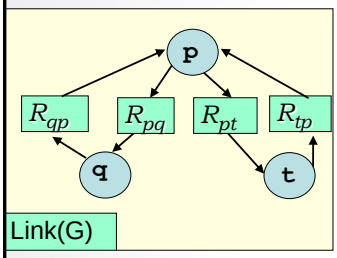


A processor *p* has a single atomic single-writer multi-reader register R_p

R_p is readable by *p*'s neighbors : *q* and *t*

R_p is writable by *p*

Communication model for network: link network model



A processor *p* has several atomic single-writer single-reader registers (one per neighbor)

R_{pt} is readable by *t*

R_{pt} is writable by *p*

Communication models based on registers on network G

Semantic / location	Atomic	Regular	Safe
State multi-reader	atomic-state(G) [MN98], [AS99], [H00], [NA02]	regular-state(G)	safe-state(G)
Link singler-reader	atomic-link(G) Read-Write atomicity model [DIM93], [Dolev 02]	regular-link(G)	safe-link(G)

Distributed System

S = (G : Graph, MC : Communication Model, A : Algo)

Computation of S is the set of computations of A on MC(G)

Goal : to implement every algorithm A for MC1(G) on MC**2(G)**

Transformation τ : MC1(G) → MC2(G)

Let *R* be a register of MC1(G) writable by *p* and readable by *q*

Transformation τ **is** two programs

τ(READ(*R*)) returns a value

τ(WRITE(*R*,*v*))

These two programs are a series of **valid** READ and WRITE operations on registers of MC**2**(G)

τ(WRITE(*R*,*v*)) invocation by *p* contains only **READ** and **WRITE** operations on registers respectively readable or writable by *p*

τ(READ(*R*)) invocation by *q* contains only **READ** and **WRITE** operations on registers respectively readable or writable bq

Simple Transformation State → Link

On *p*,

τ(STATE-WRITE(R_p,*v*)) :

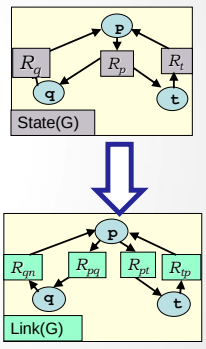
for every *q* in neighborhood of *p* do

LINK-WRITE(R_{pq}, *v*)

τ(STATE-READ(R_q))

v ← LINK-READ(R_{qp})

return *v*



Transformation τ : MC1(G) → MC2(G)

Transformation τ **is** two programs
τ(READ(*R*)) returns a value
τ(WRITE(*R*,*v*))

These two programs are a series of **valid** READ and WRITE operations on registers of MC**2**(G)

- Let *A* be an algorithm on MC1(G)
  τ(*A*) : READ(*R*) and WRITE(*R*, *v*) operation invocations in *A* are respectively replaced by two program executions τ(READ(*R*)) and τ(WRITE(*R*, *v*))

τ(*A*) is an algorithm on MC2(G)

Compiler τ : MC1(G) → MC2(G)

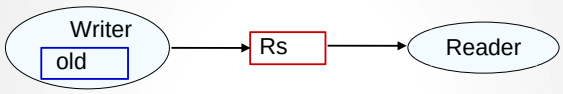
τ be a transformation of MC1(G) to MC2(G)
τ S1=(G, MC1, *A*) → S2=(G, M**C2, τ(*A*)**)

τ(*A*) : READ(*R*) and WRITE(*R*, *v*) operation invocations in *A* are respectively replaced by two program executions τ(READ(*R*)) and τ(WRITE(*R*, *v*))

τ(*A*) is an algorithm on MC2(G)

The transformation τ is a **compiler** iff S2=(G, MC2, τ(*A*)) is a syntactically and semantically valid transformation of S1 = (G, MC1, *A*)

Wait-free implementation of binary regular SWSR register by a binary safe SWSR register [Lamport 86]




```
τ(REG-WRITE(R,new))
 if old ≠ new then
    SAFE-WRITE(Rs,new))
    old ← new;
 fi
```

```
τ(REG-READ(R))
   SAFE-READ(Rs)
```


Fault tolerance : Wait-Freedom

Wait-free operation: a processor can complete the operation in a finite number of steps, regardless of the actions of other processors

(i.e. a **READ** and a **WRITE** operation is done in finite number of steps)

A wait-free operation is tolerant of processor crashes

a **compiler is wait-free** if it preserves the wait-freedom property

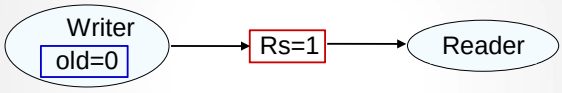
Fault-tolerance : Self-Stabilization

- Self-stabilization system automatically convergence to a legitimate configuration from any arbitrary configuration. From a legitimate configuration, the system behaves correctly (i.e. semantic of **READ** and **WRITE** operations is provided).

A self-stabilizing system is tolerant of transient failures that corrupt the processor state.

a **compiler is self-stabilizing** if it preserves the self-stabilizing property

Wait-free implementation of binary regular SWSR register by a binary safe SWSR register [Lamport 86]



All reads of the following execution return 1:
`[(regular-write(R,0), regular-read(R)]*`

Lamport construction is not self-stabilizing

Self-stabilizing, Wait-free implementation of SWSR regular binary register by a dual-reader safe binary register [Hoepman, Papatriantafilou, Tsigas 02]

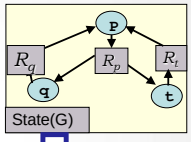

```
τ(REG-WRITE(R,new))
if SAFE-READ(Rs)≠ new
 then
    SAFE-WRITE(Rs,new))
fi
```

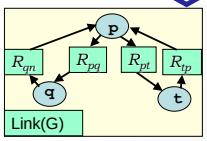
```
τ(REG-READ(R))
   SAFE-READ(Rs)
```


Simple Transformation State → Link

On p,
τ(STATE-WRITE(R_p, v)) :
  for every q in neighborhood of p do
    LINK-WRITE(R_{pq}, v)

τ(STATE-READ(R_q))
  $v \leftarrow$ LINK-READ(R_{qp})
  return v

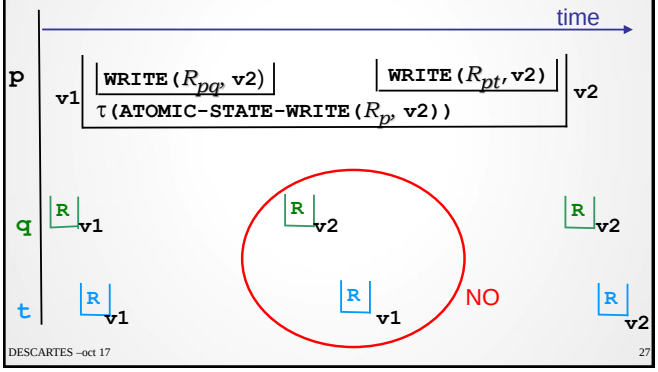




ATOMIC-LINK-WRITE(R_{pq}) ATOMIC-LINK-WRITE(R_{pt})
τ(ATOMIC-STATE-WRITE(R_p, v2))

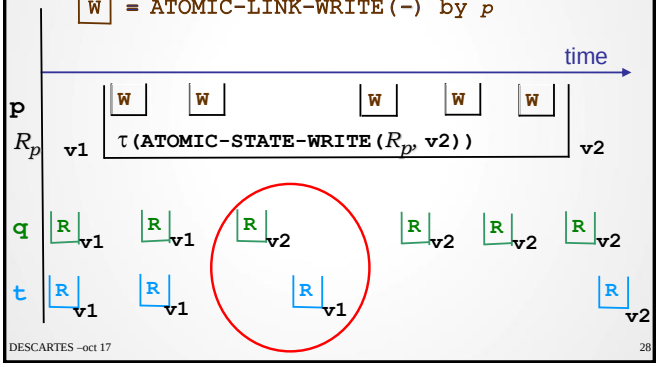
Linearization of an execution of simple transformation on atomic registers ?

Operations on R_p R = τ(ATOMIC-STATE-READ(R_p))

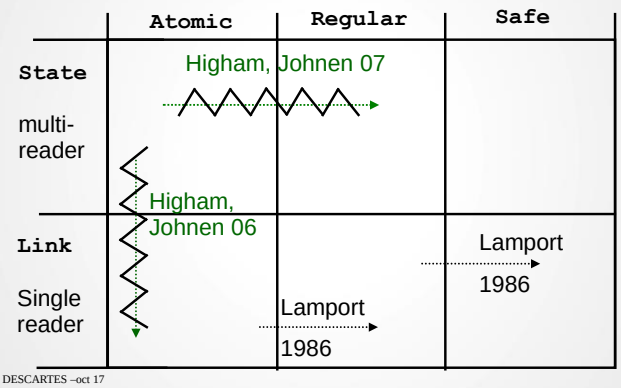


Wait-Free compiler Atomic-State(G) → Atomic-Link(G) ? NO

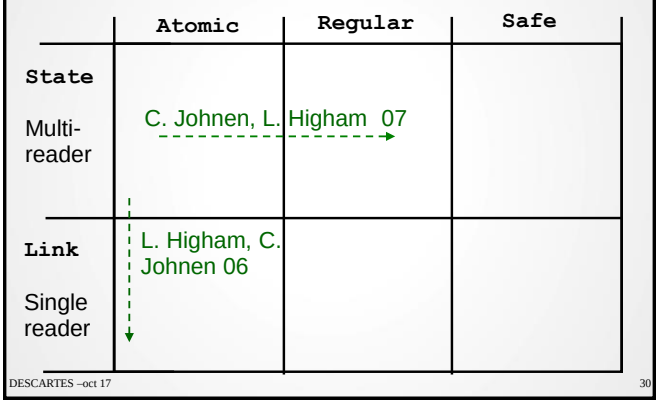
R = τ(ATOMIC-STATE-READ(R_p))
W = ATOMIC-LINK-WRITE(-) by p



Wait-free compilers



Self-stabilizing compilers



Self-Stabilizing compiler Atomic-State(G) → Atomic-Link(G) [IPDPS06]

Drawbacks/Features of Self-Stabilizing compiler from Atomic-State(G) to Atomic-Link(G) [IPDPS06] :

- τ(ATOMIC-STATE-WRITE(R_p)) is not wait-free : during an execution of τ(ATOMIC-STATE-WRITE(R_p)) any *p*'s neighbor, *q*, has to do the operation ATOMIC-LINK-READ(R_{pq}) two times
- Each process performs infinitely often ATOMIC-STATE-READ operations
- τ(ATOMIC-STATE-READ(R_p)))) is not wait-free
- Each process performs two ATOMIC-STATE-WRITE operations

Self-Stabilizing compiler Atomic-State(G) → Regular-State(G)

Drawbacks/Features of compiler from Atomic-State(G) of Regular-State(G) :

- τ(ATOMIC-STATE-WRITE(R_p)) is wait-free
- τ(ATOMIC-STATE-READ(R_p)) is not wait-free in case there is an overlay τ(ATOMIC-STATE-WRITE (R_p))
- Each process *p* performs one τ(ATOMIC-STATE-WRITE (R_p)) operation

Wait-free and Self-stabilizing compilers