

Proving Correctness of Highly-Concurrent Linearisable Objects

Viktor Vafeiadis¹, Maurice Herlihy², Tony Hoare³,
Marc Shapiro⁴

¹ University of Cambridge ² Brown University ³ Microsoft Research

⁴ INRIA Rocquencourt & LIP6

March 30, 2006

Concurrent programming is hard!

Intellectually demanding:

- Uncontrolled concurrency \Rightarrow bugs
- Global reasoning
- Interference
- All possible interleavings

Pitfalls:

- Too many conventions
- Unknown concurrency in libraries
- Trading off correctness for performance
- Deadlock, livelock, fairness, etc.

Formalise & check concurrency properties

Sequential reasoning

$\{p\} \text{ C } \{q\}$
precondition postcondition

$y := x * x$

$y := y/x$

Sequential reasoning

$\{p\} \text{ C } \{q\}$
precondition postcondition

$\{x \neq 0\}$

$y := x * x$

$y := y/x$

Sequential reasoning

$\{p\}$ C $\{q\}$
precondition postcondition

$\{x \neq 0\}$

$y := x * x$

$\{y = x^2 \wedge x \neq 0\}$

$y := y/x$

Sequential reasoning

$\{p\}$ C $\{q\}$
precondition postcondition

$\{x \neq 0\}$

$y := x * x$

$\{y = x^2 \wedge x \neq 0\}$

$y := y/x$

$\{y = x\}$

Rely/guarantee reasoning

Jones '83

$$C \models (p, R, G, q)$$

p : precondition

q : postcondition

R : rely

G : guarantee

R satisfied by all atomic actions of environment
= interference accepted from environment

G satisfied by all atomic actions of program
= interference imposed on environment

Typical rely/guarantee clauses

- $ID(x) \stackrel{\text{def}}{=} (x = \overleftarrow{x})$
 - R : no action of the environment writes x
 - G : no action of my thread writes x
- $Preserve(p) \stackrel{\text{def}}{=} (\overleftarrow{p} \Rightarrow p)$
 - R : if p true, no action of the environment makes p false
 - G : if p true, no action of my thread makes p false

R, G contain $Preserve(\textit{invariant})$

Typical rely/guarantee clauses

- $ID(x) \stackrel{\text{def}}{=} (x = \overleftarrow{x})$
 - R : no action of the environment writes x
 - G : no action of my thread writes x
- $Preserve(p) \stackrel{\text{def}}{=} (\overleftarrow{p} \Rightarrow p)$
 - R : if p true, no action of the environment makes p false
 - G : if p true, no action of my thread makes p false

R, G contain $Preserve(\textit{invariant})$

Typical rely/guarantee clauses

- $ID(x) \stackrel{\text{def}}{=} (x = \overleftarrow{x})$
 - R : no action of the environment writes x
 - G : no action of my thread writes x
- $Preserve(p) \stackrel{\text{def}}{=} (\overleftarrow{p} \Rightarrow p)$
 - R : if p true, no action of the environment makes p false
 - G : if p true, no action of my thread makes p false

R, G contain $Preserve(\textit{invariant})$

Parallel composition

- A thread must not interfere with the other(s)

$$\frac{\begin{array}{l} C_1 \models (p_1, R_1, G_1, q_1) \quad \mathbf{G}_1 \Rightarrow \mathbf{R}_2 \\ C_2 \models (p_2, R_2, G_2, q_2) \quad \mathbf{G}_2 \Rightarrow \mathbf{R}_1 \end{array}}{C_1 \parallel C_2 \models (p_1 \wedge p_2, R_1 \wedge R_2, G_1 \vee G_2, q)}$$

- Sound for safety conditions.

Linearisability

Herlihy & Wing '90

An algorithm is linearisable, if all its effects appear to take place atomically *at some instant* between the method's invocation and return.

- Execution order preserved for non-overlapping method calls
- Sequential effect of method can be described by the (p, q) of the linearisation point

Our work

Consider a ADT of integer set A with operations:

$\text{add}(e) : \langle \text{Result} := e \notin A ; A := A \cup \{e\} \rangle$

$\text{remove}(e) : \langle \text{Result} := e \in A ; A := A \setminus \{e\} \rangle$

$\text{contains}(e) : \langle \text{Result} := e \in A \rangle$

implemented as a linked-list.

Use R-G to verify several implementations:

- coarse-grain locking
- fine-grain locking
 - lock coupling
 - optimistic synchronisation
 - lazy synchronisation
- lock-free CAS-based algorithm

Our work

Consider a ADT of integer set A with operations:

$\text{add}(e) : \langle \text{Result} := e \notin A ; A := A \cup \{e\} \rangle$

$\text{remove}(e) : \langle \text{Result} := e \in A ; A := A \setminus \{e\} \rangle$

$\text{contains}(e) : \langle \text{Result} := e \in A \rangle$

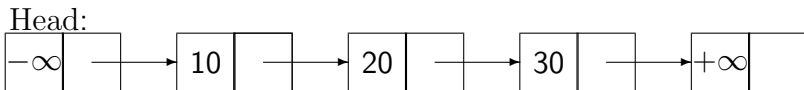
implemented as a linked-list.

Use R-G to verify several implementations:

- coarse-grain locking
- fine-grain locking
 - lock coupling
 - optimistic synchronisation
 - lazy synchronisation
- lock-free CAS-based algorithm

Lazy synchronisation list algorithm

Heller, Herlihy, Luchangco, Moir, Scherer & Shavit '05



$$A = \{10, 20, 30\}$$

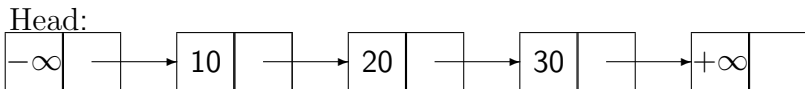
- optimistic lookup
- wait-free *contains*(*e*)

Abstraction map:

$$A = \{n.val \mid \text{Head} \rightarrow^* n \wedge \neg n.marked \wedge n.val \neq \pm\infty\}$$

Lazy synchronisation list algorithm

Heller, Herlihy, Luchangco, Moir, Scherer & Shavit '05



$$A = \{10, 20, 30\}$$

$$\begin{aligned} R &\stackrel{\text{def}}{=} \text{Preserve}(\text{ListInv}, n.\text{marked}) \wedge \text{ID}(n.\text{val}) \\ &\quad \wedge (n.\text{locked} \Rightarrow \text{ID}(n.\text{next}, n.\text{marked})) \\ &\quad \wedge (n.\text{locked} \Rightarrow \text{Preserve}(\text{Head} \rightarrow^* n)) \end{aligned}$$

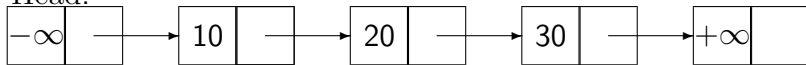
Abstraction map:

$$A = \{n.\text{val} \mid \text{Head} \rightarrow^* n \wedge \neg n.\text{marked} \wedge n.\text{val} \neq \pm\infty\}$$

Lazy synchronisation list algorithm

Heller, Herlihy, Luchangco, Moir, Scherer & Shavit '05

Head:



$$A = \{10, 20, 30\}$$

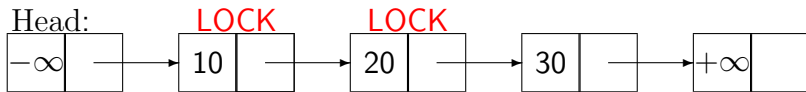
$$\begin{aligned} G \stackrel{\text{def}}{=} & \text{Preserve}(\text{ListInv}, n.\text{marked}) \wedge \text{ID}(n.\text{val}) \\ & \wedge (\neg n.\text{locked} \Rightarrow \text{ID}(n.\text{next}, n.\text{marked})) \\ & \wedge (\neg n.\text{locked} \Rightarrow \text{Preserve}(\text{Head} \rightarrow^* n)) \end{aligned}$$

Abstraction map:

$$A = \{n.\text{val} \mid \text{Head} \rightarrow^* n \wedge \neg n.\text{marked} \wedge n.\text{val} \neq \pm\infty\}$$

Lazy synchronisation list algorithm

Heller, Herlihy, Luchangco, Moir, Scherer & Shavit '05



T1: remove(20)

$A = \{10, 20, 30\}$

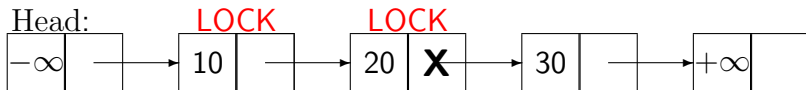
$$G \stackrel{\text{def}}{=} \text{Preserve}(\text{ListInv}, n.\text{marked}) \wedge \text{ID}(n.\text{val}) \\ \wedge (\neg n.\text{locked} \Rightarrow \text{ID}(n.\text{next}, n.\text{marked})) \\ \wedge (\neg n.\text{locked} \Rightarrow \text{Preserve}(\text{Head} \rightarrow^* n))$$

Abstraction map:

$$A = \{n.\text{val} \mid \text{Head} \rightarrow^* n \wedge \neg n.\text{marked} \wedge n.\text{val} \neq \pm\infty\}$$

Lazy synchronisation list algorithm

Heller, Herlihy, Luchangco, Moir, Scherer & Shavit '05



T1: remove(20)

$A = \{10, 30\}$

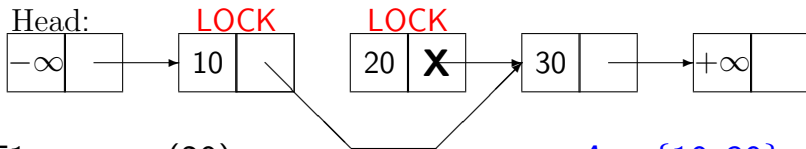
$$G \stackrel{\text{def}}{=} \text{Preserve}(\text{ListInv}, n.\text{marked}) \wedge \text{ID}(n.\text{val}) \\ \wedge (\neg n.\text{locked} \Rightarrow \text{ID}(n.\text{next}, n.\text{marked})) \\ \wedge (\neg n.\text{locked} \Rightarrow \text{Preserve}(\text{Head} \rightarrow^* n))$$

Abstraction map:

$$A = \{n.\text{val} \mid \text{Head} \rightarrow^* n \wedge \neg n.\text{marked} \wedge n.\text{val} \neq \pm\infty\}$$

Lazy synchronisation list algorithm

Heller, Herlihy, Luchangco, Moir, Scherer & Shavit '05



T1: remove(20)

$A = \{10, 30\}$

$$G \stackrel{\text{def}}{=} \text{Preserve}(\text{ListInv}, n.\text{marked}) \wedge \text{ID}(n.\text{val}) \\ \wedge (\neg n.\text{locked} \Rightarrow \text{ID}(n.\text{next}, n.\text{marked})) \\ \wedge (\neg n.\text{locked} \Rightarrow \text{Preserve}(\text{Head} \rightarrow^* n))$$

Abstraction map:

$$A = \{n.\text{val} \mid \text{Head} \rightarrow^* n \wedge \neg n.\text{marked} \wedge n.\text{val} \neq \pm\infty\}$$

Lazy synchronisation list algorithm

Heller, Herlihy, Luchangco, Moir, Scherer & Shavit '05



T1: remove(20)

$A = \{10, 30\}$

$$\begin{aligned} G \stackrel{\text{def}}{=} & \text{Preserve}(\text{ListInv}, n.\text{marked}) \wedge \text{ID}(n.\text{val}) \\ & \wedge (\neg n.\text{locked} \Rightarrow \text{ID}(n.\text{next}, n.\text{marked})) \\ & \wedge (\neg n.\text{locked} \Rightarrow \text{Preserve}(\text{Head} \rightarrow^* n)) \end{aligned}$$

Abstraction map:

$$A = \{n.\text{val} \mid \text{Head} \rightarrow^* n \wedge \neg n.\text{marked} \wedge n.\text{val} \neq \pm\infty\}$$

Lazy synchronisation list algorithm

Heller, Herlihy, Luchangco, Moir, Scherer & Shavit '05



T2: add(20)

$A = \{10, 30\}$

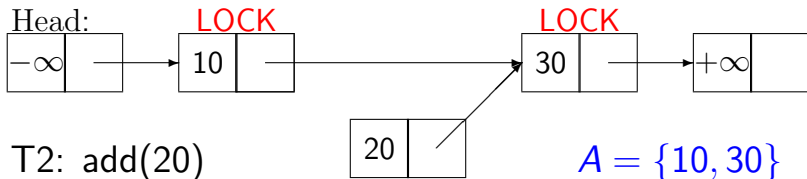
$$G \stackrel{\text{def}}{=} \text{Preserve}(\text{ListInv}, n.\text{marked}) \wedge \text{ID}(n.\text{val}) \\ \wedge (\neg n.\text{locked} \Rightarrow \text{ID}(n.\text{next}, n.\text{marked})) \\ \wedge (\neg n.\text{locked} \Rightarrow \text{Preserve}(\text{Head} \rightarrow^* n))$$

Abstraction map:

$$A = \{n.\text{val} \mid \text{Head} \rightarrow^* n \wedge \neg n.\text{marked} \wedge n.\text{val} \neq \pm\infty\}$$

Lazy synchronisation list algorithm

Heller, Herlihy, Luchangco, Moir, Scherer & Shavit '05



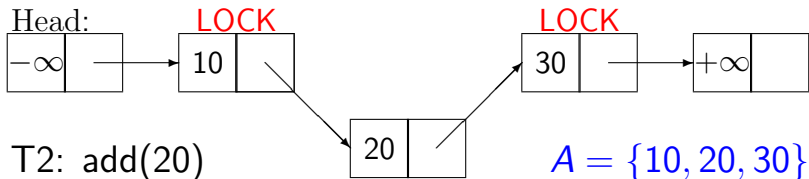
$$G \stackrel{\text{def}}{=} \text{Preserve}(\text{ListInv}, n.\text{marked}) \wedge \text{ID}(n.\text{val}) \\ \wedge (\neg n.\text{locked} \Rightarrow \text{ID}(n.\text{next}, n.\text{marked})) \\ \wedge (\neg n.\text{locked} \Rightarrow \text{Preserve}(\text{Head} \rightarrow^* n))$$

Abstraction map:

$$A = \{n.\text{val} \mid \text{Head} \rightarrow^* n \wedge \neg n.\text{marked} \wedge n.\text{val} \neq \pm\infty\}$$

Lazy synchronisation list algorithm

Heller, Herlihy, Luchangco, Moir, Scherer & Shavit '05



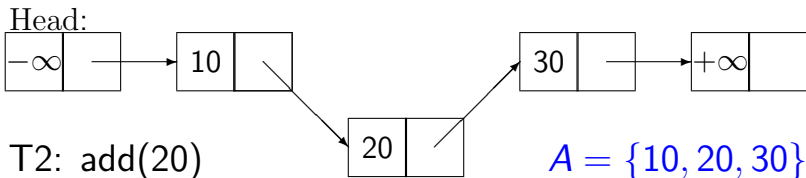
$$G \stackrel{\text{def}}{=} \text{Preserve}(\text{ListInv}, n.\text{marked}) \wedge \text{ID}(n.\text{val}) \\ \wedge (\neg n.\text{locked} \Rightarrow \text{ID}(n.\text{next}, n.\text{marked})) \\ \wedge (\neg n.\text{locked} \Rightarrow \text{Preserve}(\text{Head} \rightarrow^* n))$$

Abstraction map:

$$A = \{n.\text{val} \mid \text{Head} \rightarrow^* n \wedge \neg n.\text{marked} \wedge n.\text{val} \neq \pm\infty\}$$

Lazy synchronisation list algorithm

Heller, Herlihy, Luchangco, Moir, Scherer & Shavit '05



$$G \stackrel{\text{def}}{=} \text{Preserve}(\text{ListInv}, n.\text{marked}) \wedge \text{ID}(n.\text{val}) \\ \wedge (\neg n.\text{locked} \Rightarrow \text{ID}(n.\text{next}, n.\text{marked})) \\ \wedge (\neg n.\text{locked} \Rightarrow \text{Preserve}(\text{Head} \rightarrow^* n))$$

Abstraction map:

$$A = \{n.\text{val} \mid \text{Head} \rightarrow^* n \wedge \neg n.\text{marked} \wedge n.\text{val} \neq \pm\infty\}$$

add(e) – Insert number to the list

```
n1, n3 := locate(e) ;  
n1.lock() ; n3.lock() ;  
if  $\neg$ validate(n1, n3) then retry ...  
if n3.val  $\neq$  e then  
    n2 := new Node(e) ;  
    n2.next := n3 ;  
    n1.next := n2 ;  
  
n1.unlock() ; n3.unlock()
```

add(e) – Insert number to the list

```
n1, n3 := locate(e) ;  
n1.lock() ; n3.lock() ;  
if  $\neg$ validate(n1, n3) then retry ...  
if n3.val  $\neq$  e then  
    n2 := new Node(e) ;  
    n2.next := n3 ;  
    < n1.next := n2 ;  
    A := A  $\cup$  {e}>  
    n1.unlock() ; n3.unlock()
```

remove(e) – Delete number from the list

```
n1, n2 := locate(e) ;  
n1.lock() ; n2.lock() ;  
if  $\neg$ validate(n1, n2) then retry...  
if n2.val = e then  
    n2.marked := true ;           — logical deletion  
  
    n3 := n2.next ;  
    n1.next := n3 ;               — physical deletion  
    n1.unlock() ; n2.unlock()
```

remove(e) – Delete number from the list

```
n1, n2 := locate(e) ;  
n1.lock() ; n2.lock() ;  
if  $\neg$ validate(n1, n2) then retry...  
if n2.val = e then  
   $\langle$  n2.marked := true ;           — logical deletion  
  A := A \ {e}  $\rangle$   
  n3 := n2.next ;  
  n1.next := n3 ;                 — physical deletion  
  n1.unlock() ; n2.unlock()
```

Remarks about the proofs

R-G reasoning

- Can handle fine-grain concurrent algorithms
- Captures implicit assumptions
- Found bug, gained insight

Embedding abstract action at linearisation point

- Simple proof technique
- Easy to understand
- Does not always work [wait-free contains(e)]

Related work

R-G history

- Owicki/Gries method, '76
- Rely/guarantee [Jones '83]
- Soundness of \parallel rule [Abadi & Lamport '95]
- Hardware verification [Henzinger et al. '98]
- Automated reasoning
[Dingel '03, Flanagan et al. '05]

Other approaches:

- TLA spec & model check [Lamport]
- Simulation-based proof
[Colvin, Groves, Luchangco, Moir '06]

Conclusion

Parallel composition rule:

$$\frac{\begin{array}{l} C_1 \models (p_1, R_1, G_1, q_1) \quad \mathbf{G}_1 \Rightarrow \mathbf{R}_2 \\ C_2 \models (p_2, R_2, G_2, q_2) \quad \mathbf{G}_2 \Rightarrow \mathbf{R}_1 \end{array}}{C_1 \parallel C_2 \models (p_1 \wedge p_2, R_1 \wedge R_2, G_1 \vee G_2, q)}$$

Further work

- In practice, $R \approx G$
- Very close to sequential reasoning \Rightarrow automation
- Other algorithms, e.g. hashtables, MCAS