

Linearizability was introduced by *Herlihy* and *Wing* in their 1987 *POPL* paper “Axioms for Concurrent Objects”. It is used here as part of the snapshot algorithm example in Section 6 of the paper “Auxiliary Variables in TLA+”.

A data object, also called a state machine, executes commands from user processes. It is described by an initial state *InitObj* of the object and an operator *Apply*, where *Apply*(*i*, *cmd*, *st*) describes the output and new state of the object that results from process *i* executing command *cmd* when the object has state *st*. It is formally described by *Apply*, *InitObj*, a set *Procs* of processes, sets *Commands*(*i*) and *Outputs*(*i*) of possible commands and possible outputs for each process *i*, a set *ObjValues* of all possible states of the object, and an initial process *i* output value *InitOutput*(*i*) whose use is explained below. Here are the declarations of these constants and constant operators and the property they are assumed to satisfy. The requirement that *Outputs*(*i*) and *Commands*(*i*) are disjoint sets is not essential, but makes our specification of linearizability simpler.

22 CONSTANTS *Procs*, *Commands*($_$), *Outputs*($_$), *InitOutput*($_$),
 23 *ObjValues*, *InitObj*, *Apply*($_$, $_$, $_$)

25 ASSUME *LinearAssumps* \triangleq
 26 \wedge *InitObj* \in *ObjValues*
 27 $\wedge \forall i \in \textit{Procs} : \textit{InitOutput}(i) \in \textit{Outputs}(i)$
 28 $\wedge \forall i \in \textit{Procs} : \textit{Outputs}(i) \cap \textit{Commands}(i) = \{\}$
 29 $\wedge \forall i \in \textit{Procs}, obj \in \textit{ObjValues} :$
 30 $\quad \forall cmd \in \textit{Commands}(i) :$
 31 $\quad \wedge \textit{Apply}(i, cmd, obj).output \in \textit{Outputs}(i)$
 32 $\quad \wedge \textit{Apply}(i, cmd, obj).newState \in \textit{ObjValues}$

A linearizable implementation of a data object is one in which each operation execution by a process *i* is performed by three steps:

- An externally visible *BeginOp*(*i*, *cmd*) step that begins an operation with command *cmd*.
- An internal *DoOp*(*i*) step that performs the operation on the object and obtains the output.
- An externally visible *EndOp*(*i*) step that displays the output.

These actions are described with three variables:

object

An internal variable whose value is the current state of the object.

interface

An externally visible variable whose value is a function with domain *Procs* that describes the issuing of commands and the return of values. The value of *interface*[*i*] is initially *InitOutput*(*i*), it is set to *cmd* by the *BeginOp*(*i*, *cmd*) step, and is set to the command's output by the *EndOp*(*i*) step.

istate

An internal variable whose value is a function with domain *Procs*. The element *istate*[*i*] is used to remember, when process *i* is executing a command, if it has executed the *DoOp*(*i*) step and, if so, what the output value is. It initially equals *InitOutput*(*i*), is set to *cmd* by a *BeginOp*(*i*, *cmd*) step, and is set to the output value by a *DoOp*(*i*) step.

The definitions of the initial predicate and next-state action should be easy to understand.

70 VARIABLES *object*, *interface*, *istate*
 71 vars $\triangleq \langle \textit{object}, \textit{interface}, \textit{istate} \rangle$

```

73  $TypeOK \triangleq \wedge object \in ObjValues$ 
74  $\wedge \forall i \in Procs : \wedge interface[i] \in Outputs(i) \cup Commands(i)$ 
75  $\wedge istate[i] \in Outputs(i) \cup Commands(i)$ 

77  $Init \triangleq \wedge object = InitObj$ 
78  $\wedge interface = [i \in Procs \mapsto InitOutput(i)]$ 
79  $\wedge istate = [i \in Procs \mapsto InitOutput(i)]$ 

81  $BeginOp(i, cmd) \triangleq \wedge interface[i] \in Outputs(i)$ 
82  $\wedge interface' = [interface \text{ EXCEPT } ![i] = cmd]$ 
83  $\wedge istate' = [istate \text{ EXCEPT } ![i] = cmd]$ 
84  $\wedge object' = object$ 

86  $DoOp(i) \triangleq \wedge interface[i] \in Commands(i)$ 
87  $\wedge istate[i] = interface[i]$ 
88  $\wedge LET \text{ result} \triangleq Apply(i, interface[i], object)$ 
89  $IN \wedge object' = result.newState$ 
90  $\wedge istate' = [istate \text{ EXCEPT } ![i] = result.output]$ 
91  $\wedge interface' = interface$ 

93  $EndOp(i) \triangleq \wedge interface[i] \in Commands(i)$ 
94  $\wedge istate[i] \in Outputs(i)$ 
95  $\wedge interface' = [interface \text{ EXCEPT } ![i] = istate[i]]$ 
96  $\wedge UNCHANGED \langle object, istate \rangle$ 

98  $Next \triangleq \exists i \in Procs : \vee \exists cmd \in Commands(i) : BeginOp(i, cmd)$ 
99  $\vee DoOp(i)$ 
100  $\vee EndOp(i)$ 

```

For later use, we give a name to the safety part of the spec.

```

105  $SafeSpec \triangleq Init \wedge \Box[Next]_{vars}$ 

```

The liveness requirement is that an operation that has begun (by executing a $BeginOp(i, cmd)$ step) must eventually finish (by executing an $EndOp(i)$ step). This is expressed by the formulas *Fairness* defined below. The equivalent specification is obtained by defining *Fairness* to equal this formula:

```

 $\forall i \in Procs : WF_{vars}(DoOp(i) \vee EndOp(i))$ 

```

```

116  $Fairness \triangleq \forall i \in Procs : WF_{vars}(DoOp(i)) \wedge WF_{vars}(EndOp(i))$ 
117  $Spec \triangleq SafeSpec \wedge Fairness$ 

```

118

```

\ * Modification History
\ * Last modified Sat Oct 22 01:28:13 PDT 2016 by lamport
\ * Created Tue Oct 04 02:01:02 PDT 2016 by lamport

```