MODULE *Linearizability*

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.

$$
\begin{array}{l}
\textsc{constants}\quad Procs,\ Commands(\_),\ Outputs(\_),\ InitOutput(\_),\\
\qquad\qquad\qquad ObjValues,\ InitObj,\ Apply(\_, \_, \_)
\end{array}
$$

$$
\begin{array}{l}
\textsc{assume}\ LinearAssumps \triangleq\\
\qquad \land\ InitObj \in ObjValues\\
\qquad \land\ \forall i \in Procs : InitOutput(i) \in Outputs(i)\\
\qquad \land\ \forall i \in Procs : Outputs(i) \cap Commands(i) = \{\}\\
\qquad \land\ \forall i \in Procs,\ obj \in ObjValues :\\
\qquad\qquad \forall cmd \in Commands(i) :\\
\qquad\qquad\qquad \land\ Apply(i, cmd, obj).output \in Outputs(i)\\
\qquad\qquad\qquad \land\ Apply(i, cmd, obj).newState \in ObjValues
\end{array}
$$

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.

$$
\begin{array}{l}
\textsc{variables}\ object,\ interface,\ istate\\
vars \triangleq \langle object, interface, istate \rangle
\end{array}
$$

$$
\begin{array}{l}
TypeOK \triangleq \land\ object \in ObjValues \\
\qquad\qquad\quad \land\ \forall\, i \in Procs :\ \land\ interface[i] \in Outputs(i) \cup Commands(i) \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\ istate[i] \in Outputs(i) \cup Commands(i)
\end{array}
$$

$$
\begin{array}{l}
Init \triangleq \land\ object\ = InitObj \\
\qquad\quad\ \land\ interface = [i \in Procs \mapsto InitOutput(i)] \\
\qquad\quad\ \land\ istate = [i \in Procs \mapsto InitOutput(i)]
\end{array}
$$

$$
\begin{array}{l}
BeginOp(i,\ cmd) \triangleq \land\ interface[i] \in Outputs(i) \\
\qquad\qquad\qquad\qquad\ \land\ interface' = [interface\ \text{EXCEPT}\ ![i] = cmd] \\
\qquad\qquad\qquad\qquad\ \land\ istate' = [istate\ \text{EXCEPT}\ ![i] = cmd] \\
\qquad\qquad\qquad\qquad\ \land\ object' = object
\end{array}
$$

$$
\begin{array}{l}
DoOp(i) \triangleq \land\ interface[i] \in Commands(i) \\
\qquad\qquad\quad \land\ istate[i] = interface[i] \\
\qquad\qquad\quad \land\ \text{LET}\ result \triangleq Apply(i,\ interface[i],\ object) \\
\qquad\qquad\qquad\ \text{IN}\ \ \land\ object' = result.newState \\
\qquad\qquad\qquad\qquad\ \ \land\ istate' = [istate\ \text{EXCEPT}\ ![i] = result.output] \\
\qquad\qquad\quad \land\ interface' = interface
\end{array}
$$

$$
\begin{array}{l}
EndOp(i) \triangleq \land\ interface[i] \in Commands(i) \\
\qquad\qquad\quad\ \land\ istate[i] \in Outputs(i) \\
\qquad\qquad\quad\ \land\ interface' = [interface\ \text{EXCEPT}\ ![i] = istate[i]] \\
\qquad\qquad\quad\ \land\ \text{UNCHANGED}\ \langle object,\ istate \rangle
\end{array}
$$

$$
\begin{array}{l}
Next \triangleq \exists\, i \in Procs :\ \lor\ \exists\, cmd \in Commands(i) : BeginOp(i,\ cmd) \\
\qquad\qquad\qquad\qquad\quad\ \lor\ DoOp(i) \\
\qquad\qquad\qquad\qquad\quad\ \lor\ EndOp(i)
\end{array}
$$

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

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

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*$(DoOp(i) \lor EndOp(i))$

$$
\begin{array}{l}
Fairness \triangleq \forall\, i \in Procs : \mathrm{WF}_{vars}(DoOp(i)) \land \mathrm{WF}_{vars}(EndOp(i)) \\
Spec \triangleq SafeSpec \land Fairness
\end{array}
$$

---

* Modification History
* Last modified *Thu Nov* 08 17:15:41 *CST* 2018 by *hengxin*
* Last modified Sat *Oct* 22 01:28:13 *PDT* 2016 by *lamport*
* Created *Tue Oct* 04 02:01:02 *PDT* 2016 by *lamport*