MODULE *NewLinearSnapshot*

This module is part of the *AfekSimplified* example in Section 6 of the paper "Auxiliary Variables in TLA+". It is equivalent to the specification in module *LinearSnapshot* of a linearizable snapshot algorithm, in the sense that it permits the same behaviors of the externally visible variable *interface* . You should understand that specification before reading this module.

This specification differs from the one in *LinearSnapshot* by deferring the choice of a reader's output as long as possible–namely, the choice is made only in the *EndRd* action. The reader maintains a list of all the values that the memory *mem* had while the read operation is being performed, and has the *EndRd* action choose an arbitrary element of that list to return as the output.

The equivalence of the two linearizable snapshot algorithms means that if *Spec_NL* is formula *Spec* of this module and *Spec_L* is formula *Spec* of module *LinearSnapshot*, then $\exists\, mem, rstate, wstate : Spec\_NL$ is equivalent to $\exists\, mem, istate : Spec\_L$. We only show that the first implies the second, since that is all we need for our example of the simplified *Afek* et al. snapshot algorithm.

EXTENDS *Integers*, *Sequences*

The declared and defined constants are the same as in module *LinearSnapshot*.

CONSTANTS *Readers*, *Writers*, *RegVals*, *InitRegVal*

ASSUME $\land Readers \cap Writers = \{\}$
$\land InitRegVal \in RegVals$

$InitMem \triangleq [i \in Writers \mapsto InitRegVal]$
$MemVals \triangleq [Writers \to RegVals]$
$NotMemVal \triangleq$ CHOOSE $v : v \notin MemVals$
$NotRegVal \triangleq$ CHOOSE $v : v \notin RegVals$

The variables *mem* and *interface* are the same as in *LinearSnapshot*. The variable *wstate* is a function with domain *Writers*, where $wstate[i]$ assumes the same value as $istate[i]$ does for the *LinearSnapshot* spec, for all $i \in Writers$. The variable *rstate* is a function with domain *Readers* such that $rstate[i] = \langle\rangle$ when reader i is not performing a read operation, and while performing a read it equals the sequence of values that *mem* has assumed since the $BeginRd(i)$ step.

We will not bother explaining the assertions that the spec makes about *mem*, *interface* , and *wstate*, since they are exactly the same as in *LinearSnapshot* for *mem* and *interface* , and every condition on $wstate[i]$ is the same as the corresponding condition on $istate[i]$ in *LinearSnapshot*, for $i \in Writers$.

VARIABLES *mem*, *interface*, *rstate*, *wstate*
$vars \triangleq \langle mem, interface, rstate, wstate\rangle$

$$
\begin{array}{ll}
TypeOK \triangleq & \land\ mem \in MemVals \\
& \land\ \land \text{DOMAIN } interface = Readers \cup Writers \\
& \quad \land \forall\, i \in Readers : interface[i] \in MemVals \cup \{NotMemVal\} \\
& \quad \land \forall\, i \in Writers : interface[i] \in RegVals \ \cup \{NotRegVal\} \\
& \land\ \land\ rstate \in [Readers \to Seq(MemVals)] \\
& \quad \land \forall\, i \in Readers : \\
& \qquad (rstate[i] = \langle\rangle) \equiv (interface[i] \in MemVals) \\
& \land\ wstate \in [Writers \to RegVals \cup \{NotRegVal\}]
\end{array}
$$

Since no reader is initially executing a read command, $rstate[i]$ initially equals the empty sequence for each reader i.

$$
\begin{aligned}
Init \triangleq\ & \land\ mem = InitMem \\
& \land\ interface = [i \in Readers \cup Writers \mapsto \\
& \qquad \text{IF } i \in Readers \text{ THEN } InitMem \text{ ELSE } NotRegVal] \\
& \land\ rstate\ = [i \in Readers \mapsto \langle\rangle] \\
& \land\ wstate = [i \in Writers \mapsto NotRegVal]
\end{aligned}
$$

Since they leave $rstate$ unchanged, $BeginWr$ and $EndWr$ are the same as in $LinearSnapshot$.

$$
\begin{aligned}
BeginWr(i,\ cmd) \triangleq\ & \land\ interface[i] = NotRegVal \\
& \land\ interface' = [interface \text{ EXCEPT } ![i] = cmd] \\
& \land\ wstate' = [wstate \text{ EXCEPT } ![i] = cmd] \\
& \land \text{UNCHANGED } \langle mem,\ rstate\rangle
\end{aligned}
$$

The $BeginRd(i)$ action sets $rstate[i]$ to $\langle mem\rangle$, since the current value of mem is the only output value $EndRd(i)$ can return if executed immediately afterwards.

$$
\begin{aligned}
BeginRd(i) \triangleq\ & \land\ interface[i] \in MemVals \\
& \land\ interface' = [interface \text{ EXCEPT } ![i] = NotMemVal] \\
& \land\ rstate' = [rstate \text{ EXCEPT } ![i] = \langle mem\rangle] \\
& \land \text{UNCHANGED } \langle mem,\ wstate\rangle
\end{aligned}
$$

The $DoWr(i)$ action appends the new value of mem to $rstate[j]$ for each reader j currently performing a read operation, since each of those readers can return that value as their output.

$$
\begin{aligned}
DoWr(i) \triangleq\ & \land\ interface[i] \in RegVals \\
& \land\ wstate[i] = interface[i] \\
& \land\ mem' = [mem \text{ EXCEPT } ![i] = interface[i]] \\
& \land\ wstate' = [wstate \text{ EXCEPT } ![i] = NotRegVal] \\
& \land\ rstate'\ = [j \in Readers \mapsto \\
& \qquad \text{IF } rstate[j] = \langle\rangle \\
& \qquad\quad \text{THEN } \langle\rangle \\
& \qquad\quad \text{ELSE } Append(rstate[j],\ mem')] \\
& \land\ interface' = interface
\end{aligned}
$$

$EndRd(i)$ can set as its output any element of $rstate[i]$. It resets $rstate[i]$ to the empty sequence.

$$
\begin{aligned}
EndRd(i) \triangleq\ & \land\ interface[i] = NotMemVal \\
& \land\ \exists\, j \in 1\,..\,Len(rstate[i]) : \\
& \qquad interface' = [interface \text{ EXCEPT } ![i] = rstate[i][j]] \\
& \land\ rstate' = [rstate \text{ EXCEPT } ![i] = \langle\rangle] \\
& \land \text{UNCHANGED } \langle mem,\ wstate\rangle
\end{aligned}
$$

$$
\begin{aligned}
EndWr(i) \triangleq\ & \land\ interface[i] \in RegVals \\
& \land\ wstate[i] = NotRegVal \\
& \land\ interface' = [interface \text{ EXCEPT } ![i] = wstate[i]] \\
& \land \text{UNCHANGED } \langle mem,\ rstate,\ wstate\rangle
\end{aligned}
$$

$$
Next \triangleq\ \lor\, \exists\, i \in Readers : BeginRd(i) \lor EndRd(i)
$$

$$\lor \exists\, i \in \mathit{Writers} : \lor \exists\, \mathit{cmd} \in \mathit{RegVals} : \mathit{BeginWr}(i,\ \mathit{cmd})$$
$$\lor\ \mathit{DoWr}(i) \lor \mathit{EndWr}(i)$$

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

The fairness condition implies that every read or write operation that has begun must eventually finish.

$$\mathit{Fairness} \triangleq \land \forall\, i \in \mathit{Readers} : \mathrm{WF}_{\mathit{vars}}(\mathit{EndRd}(i))$$
$$\land \forall\, i \in \mathit{Writers} : \mathrm{WF}_{\mathit{vars}}(\mathit{DoWr}(i)) \land \mathrm{WF}_{\mathit{vars}}(\mathit{EndWr}(i))$$

$\mathit{Spec} \triangleq \mathit{Init} \land \Box[\mathit{Next}]_{\mathit{vars}} \land \mathit{Fairness}$

* Modification History
* Last modified Sat *Oct* 22 01:41:33 *PDT* 2016 by *lamport*
* Created *Wed Oct* 05 01:23:41 *PDT* 2016 by *lamport*