# MODULE *AfekSimplified*

This module specifies the simplified *Afek* et al. snapshot algorithm algorithm described in Section 6.3 of the paper "Auxiliary Variables in TLA+". This is a simplified version of an algorithm in the 1993 paper "Atomic snapshots of Shared Memory" by *Afek*, *Attiya*, *Dolev*, *Gafni*, Merritt, and *Shavit*. It will be shown to satisfy the safety specification of a linearizable snapshot object in module *NewLinearSnapshot*. (The actual algorithm by *Afek* et al. also satisfies the specification's liveness property, but our simplified version does not.)

EXTENDS *Integers*

We begin by declaring and defining the same constants as in module *NewLinearSnapshot*.

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

$$\begin{array}{ll}\textsc{assume} & \land\ Readers \cap Writers = \{\} \\ & \land\ InitRegVal \in RegVals\end{array}$$

$$\begin{array}{lcl}MemVals & \triangleq & [Writers \rightarrow RegVals] \\ InitMem & \triangleq & [i \in Writers \mapsto InitRegVal] \\ NotMemVal & \triangleq & \textsc{choose}\ v : v \notin MemVals \\ NotRegVal & \triangleq & \textsc{choose}\ v : v \notin RegVals\end{array}$$

Instead of the internal variable mem of the specification, the algorithm maintains an internal variable *imem* such that for each writer $i$, the value of $imem[i]$ is a pair $\langle v, k \rangle$, where $v$ is the last register value written by $i$, and $k$ is the number of times the register has been written by $i$. The purpose of the second component of $imem[i]$ is to ensure that values written to $imem[i]$ by writer $i$ in different write operations are different.

We now define some constants, including the set *IMemVals* of all possible values of *imem*.

$$\begin{array}{lcl}IRegVals & \triangleq & RegVals \times Nat \\ IMemVals & \triangleq & [Writers \rightarrow IRegVals] \\ InitIMem & \triangleq & [i \in Writers \mapsto \langle InitRegVal, 0 \rangle]\end{array}$$

In addition to *imem*, the algorithm has three internal variables: *wrNum*, *rdVal*1, and *rdVal*2. Each writer $i$ records in $wrNum[i]$ the number of times it has written $imem[i]$. Writer $i$ acts pretty much like the writer in the specification, except that $DoWr(i)$ writes a pair of values in $imem[i]$ and increments $wrNum[i]$. The writer needs no other internal information because it knows that it has performed a $BeginWr(i, cmd)$ step but not the subsequent $DoWr(i)$ step if $wrNum[i]$ is different from $imem[i][2]$; and it doesn't have to remember the command *cmd* because that's in $interface[i]$.

Reader $i$ keeps performing the following scan procedure until the procedure succeeds in computing an output, whereupon the read operation terminates by producing that output. The scan procedure reads *imem* by reading the elements $imem[j]$ one at a time in any order, and it then reads *imem* again by reading its elements in any order. The scan procedure succeeds if both reads obtain the same value of *imem*, in which case it produces the output consisting of the register values of that value of *imem*. (This procedure produces a correct output only because a writer $j$ cannot write the same value twice in $imem[j]$.) It's possible for the scan procedure never to succeed, in which case the read operation never terminates. Afek et al. have a method for terminating after a finite number of unsuccessful scans, but it complicates the algorithm without significantly changing the structure of its correctness proof.

Reader $i$ keeps in $rdVal1[i][j]$ and $rdVal2[i][j]$ the values of $imem[j]$ that it has read so far during the current scan procedure's reads of *imem*. The values of $rdVal1[i]$ and $rdVal2[i]$ are each a function that maps a subset of the writers to the values it has read for those writer's registers. They both equal the function $\langle\rangle$ with empty domain when the writer is not performing a scan.

With this explanation of how the algorithm works, it should be easy to understand its TLA+ specification.

$$\textsc{variables}\ imem,\ interface,\ wrNum,\ rdVal1,\ rdVal2$$
$$vars \triangleq \langle imem,\ interface,\ wrNum,\ rdVal1,\ rdVal2\rangle$$

We define $PartialFcns(U,\ V)$ to be the set of functions from a subset of $U$ to $V$. It is used only in the type-correctness invariant.

$$PartialFcns(U,\ V) \triangleq \textsc{union}\ \{[D \rightarrow V] : D \in \textsc{subset}\ U\}$$

$$\begin{aligned} TypeOK \triangleq\ & \land\ imem \in IMemVals \\ & \land\ \land\ \textsc{domain}\ interface = Readers \cup Writers \\ & \quad\ \land\ \forall\, i \in Readers : interface[i] \in MemVals \cup \{NotMemVal\} \\ & \quad\ \land\ \forall\, i \in Writers : interface[i] \in RegVals\ \ \cup \{NotRegVal\} \\ & \land\ wrNum \in [Writers \rightarrow Nat] \\ & \land\ rdVal1 \in [Readers \rightarrow PartialFcns(Writers,\ IRegVals)] \\ & \land\ rdVal2 \in [Readers \rightarrow PartialFcns(Writers,\ IRegVals)] \end{aligned}$$

$$\begin{aligned} Init \triangleq\ & \land\ imem = InitIMem \\ & \land\ interface = [i \in Readers \cup Writers \mapsto \\ & \qquad\qquad \textsc{if}\ i \in Readers\ \textsc{then}\ InitMem\ \textsc{else}\ \ NotRegVal] \\ & \land\ wrNum = [i \in Writers \mapsto 0] \\ & \land\ rdVal1 = [i \in Readers \mapsto \langle\rangle] \\ & \land\ rdVal2 = [i \in Readers \mapsto \langle\rangle] \end{aligned}$$

$$\begin{aligned} BeginWr(i,\ cmd) \triangleq\ & \land\ interface[i] = NotRegVal \\ & \land\ wrNum' = [wrNum\ \textsc{except}\ ![i] = wrNum[i] + 1] \\ & \land\ interface' = [interface\ \textsc{except}\ ![i] = cmd] \\ & \land\ \textsc{unchanged}\ \langle imem,\ rdVal1,\ rdVal2\rangle \end{aligned}$$

$$\begin{aligned} DoWr(i) \triangleq\ & \land\ interface[i] \in RegVals \\ & \land\ imem[i][2] \neq wrNum[i] \\ & \land\ imem' = [imem\ \textsc{except}\ ![i] = \langle interface[i],\ wrNum[i]\rangle] \\ & \land\ \textsc{unchanged}\ \langle interface,\ wrNum,\ rdVal1,\ rdVal2\rangle \end{aligned}$$

$$\begin{aligned} EndWr(i) \triangleq\ & \land\ interface[i] \in RegVals \\ & \land\ imem[i][2] = wrNum[i] \\ & \land\ interface'\ \ = [interface\ \textsc{except}\ ![i] = NotRegVal] \\ & \land\ \textsc{unchanged}\ \langle imem,\ wrNum,\ rdVal1,\ rdVal2\rangle \end{aligned}$$

$$\begin{aligned} BeginRd(i) \triangleq\ & \land\ interface[i] \in MemVals \\ & \land\ interface' = [interface\ \textsc{except}\ ![i] = NotMemVal] \\ & \land\ \textsc{unchanged}\ \langle imem,\ wrNum,\ rdVal1,\ rdVal2\rangle \end{aligned}$$

If $x$ is not in the domain of the function $v$, then $AddToFcn(f, x, v)$ is the function obtained from $f$ by adding $x$ to its domain and letting $x$ be mapped to $v$. This could be written as as $f$ @@ $(x :> v)$, where the operators :> and @@ are defined in the standard *TLC* module.

$$
\begin{array}{l}
AddToFcn(f,\, x,\, v) \;\triangleq \\
\quad [y \in (\textsc{domain}\ f) \cup \{x\} \mapsto \textsc{if}\ y = x\ \textsc{then}\ v\ \textsc{else}\ f[y]]
\end{array}
$$

$$
\begin{array}{ll}
Rd1(i) \;\triangleq & \land\ interface[i] = NotMemVal \\
 & \land\ \exists\, j \in Writers \setminus \textsc{domain}\ rdVal1[i] : \\
 & \qquad rdVal1' = [rdVal1\ \textsc{except}\ ![i] = AddToFcn(rdVal1[i],\, j,\, imem[j])] \\
 & \land\ \textsc{unchanged}\ \langle interface,\, imem,\, wrNum,\, rdVal2 \rangle
\end{array}
$$

$$
\begin{array}{ll}
Rd2(i) \;\triangleq & \land\ interface[i] = NotMemVal \\
 & \land\ \textsc{domain}\ rdVal1[i] = Writers \\
 & \land\ \exists\, j \in Writers \setminus \textsc{domain}\ rdVal2[i] : \\
 & \qquad rdVal2' = [rdVal2\ \textsc{except}\ ![i] = AddToFcn(rdVal2[i],\, j,\, imem[j])] \\
 & \land\ \textsc{unchanged}\ \langle interface,\, imem,\, wrNum,\, rdVal1 \rangle
\end{array}
$$

$$
\begin{array}{ll}
TryEndRd(i) \;\triangleq & \land\ interface[i] = NotMemVal \\
 & \land\ \textsc{domain}\ rdVal1[i] = Writers \\
 & \land\ \textsc{domain}\ rdVal2[i] = Writers \\
 & \land\ \textsc{if}\ rdVal1[i] = rdVal2[i] \\
 & \qquad \textsc{then}\ \ \land\ interface' = \\
 & \qquad\qquad\qquad [interface\ \textsc{except} \\
 & \qquad\qquad\qquad\quad ![i] = [j \in Writers \mapsto rdVal1[i][j][1]]] \\
 & \qquad \textsc{else}\ \ \ \land\ interface' = interface \\
 & \land\ rdVal1' = [rdVal1\ \textsc{except}\ ![i] = \langle\rangle] \\
 & \land\ rdVal2' = [rdVal2\ \textsc{except}\ ![i] = \langle\rangle] \\
 & \land\ \textsc{unchanged}\ \langle imem,\, wrNum \rangle
\end{array}
$$

$$
\begin{array}{ll}
Next \;\triangleq & \lor\ \exists\, i \in Readers : BeginRd(i) \lor Rd1(i) \lor Rd2(i) \lor TryEndRd(i) \\
 & \lor\ \exists\, i \in Writers :\ \lor\ \exists\, cmd \in RegVals : BeginWr(i,\, cmd) \\
 & \qquad\qquad\qquad\quad\ \lor\ DoWr(i) \lor EndWr(i)
\end{array}
$$

Since a read need never terminate, the algorithm doesn't satisfy the *NewLinearSnapshot* specification's liveness requirements, so we don't bother specifying any fairness of the actions.

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

* Modification History

* Last modified Sat *Oct* 22 01:58:50 *PDT* 2016 by *lamport*

* Created *Wed* *Oct* 05 08:23:18 *PDT* 2016 by *lamport*