MODULE *SimpleRegular*

This is a minor modification of the algorithm in module Simple. That algorithm is an N-process algorithm shared-memory algorithm, in which each process i has a shared register $x[i]$ that it writes and is read by process $x[(i-1)\%N]$. Each process i also has a local register $y[i]$ that only it can access.

The shared registers $x[i]$ in the algorithm of module Simple are assumed to be atomic, effectively meaning that each read or write by any process is an atomic action. In the algorithm in this module, the $x[i]$ are assumed to be a weaker class of registers called regular registers. Atomic and regular registers are defined in the paper

On Interprocess Communication Distributed Computing 1, 2 (1986), $77-101$

which can be found on the Web at

http://*lamport.azurewebsites.net*/pubs/*interprocess.pdf*

That paper considers only registers that can be written by a single process, but takes into account that reads and writes are not instantaneous atomic actions, but take a finite length of time and can overlap. An atomic register is one in which a read and write acts as if it were executed atomically at some time between the beginning and end of the operation. An atomic register can be modeled as one in which each read and write is a single step in an execution.

A regular register is defined there to be one in which a read that overlaps some (possibly empty) set of writes to a register obtains a value that is either the register's value before any of the writes were begun or one of the values being written by one of the writes that the read overlaps. (Hence, a read that overlaps no writes obtains the last value written before the read, or the initial value if there were no such writes before the read.) A regular register r can be modeled in a TLA+ spec modeled as a variable rv that equals a set of values. The register having a value v is modeled by rv equaling $\{v\}$. When a value w different from v is written to r, the value of rv first changes to $\{v, w\}$ and then to $\{w\}$. A read of r is modeled as an atomic step that can obtain any value in the set rv.

The algorithm of this model is obtained from that of module Simple by letting each value $x[i]$ be the set of values representing a regular register. Since each $y[i]$ is local to process i, we can consider it to be atomic.

The problem of generalizing the algorithm of module Simple to use regular registers was proposed by *Yuri Abraham* in

On *Lamport*'s "Teaching Concurrency" Bulletin of *EATCS* (European Association for Theoretical Computer
Science) No. 127, *February* 2019
http://*bulletin.eatcs.org*/*index.php*/beatcs/article/view/569

EXTENDS *Integers*, *TLAPS*

CONSTANT N
ASSUME $NAssump \triangleq (N \in Nat) \land (N > 0)$

```
*************************************************************************
--algorithm SimpleRegular{
    variables $x = [i \in 0 \,..\, (N-1) \mapsto \{0\}],\ y = [i \in 0 \,..\, (N-1) \mapsto 0]$ ;
    process ( $proc \in 0 \,..\, N-1$ ) {
      $a1$: $x[self] := \{0, 1\}$ ;
      $a2$: $x[self] := \{1\}$ ;
      b:  with ( $v \in x[(self - 1)\%N]$ ) { $y[self] := v$ }
    }
}
```

****

BEGIN TRANSLATION

VARIABLES $x,\ y,\ pc$

$$vars \triangleq \langle x,\ y,\ pc \rangle$$

$$ProcSet \triangleq (0 \,..\, N-1)$$

$$Init \triangleq \text{Global variables} \quad \begin{array}{l} \land\ x = [i \in 0 \,..\, (N-1) \mapsto \{0\}] \\ \land\ y = [i \in 0 \,..\, (N-1) \mapsto 0] \\ \land\ pc = [self \in ProcSet \mapsto \text{“a1”}] \end{array}$$

$$a1(self) \triangleq \begin{array}{l} \land\ pc[self] = \text{“a1”} \\ \land\ x' = [x \text{ EXCEPT } ![self] = \{0, 1\}] \\ \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“a2”}] \\ \land\ y' = y \end{array}$$

$$a2(self) \triangleq \begin{array}{l} \land\ pc[self] = \text{“a2”} \\ \land\ x' = [x \text{ EXCEPT } ![self] = \{1\}] \\ \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“b”}] \\ \land\ y' = y \end{array}$$

$$b(self) \triangleq \begin{array}{l} \land\ pc[self] = \text{“b”} \\ \land\ \exists\, v \ \in x[(self-1)\%N] : \\ \qquad y' = [y \text{ EXCEPT } ![self] = v] \\ \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“Done”}] \\ \land\ x' = x \end{array}$$

$$proc(self) \triangleq a1(self) \lor a2(self) \lor b(self)$$

Allow infinite stuttering to prevent deadlock on termination.

$$Terminating \triangleq \begin{array}{l} \land\ \forall\, self \in ProcSet : pc[self] = \text{“Done”} \\ \land\ \text{UNCHANGED } vars \end{array}$$

$$Next \triangleq \begin{array}{l} (\exists\, self \in 0 \,..\, N-1 : proc(self)) \\ \quad \lor\ Terminating \end{array}$$

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

$$Termination \triangleq \Diamond(\forall\, self \in ProcSet : pc[self] = \text{“Done”})$$

END TRANSLATION

The definition of *PCorrect* is the same as in module Simple.

$$\begin{array}{l} \mathit{PCorrect} \triangleq (\forall\, i \in 0 \,..\, (N-1) : pc[i] = \text{"Done"}) \Rightarrow \\ \qquad\qquad\quad (\exists\, i \in 0 \,..\, (N-1) : y[i] = 1) \end{array}$$

The type invariant *TypeOK* is the obvious modification of the type invariant *TypeOK* of module Simple. Except for the change to the definition of *TypeOK*, the inductive invariant *Inv* is the same as in module Simple.

$$\begin{array}{lll} \mathit{TypeOK} & \triangleq & \land\; x \in [0 \,..\, (N-1) \to (\text{SUBSET } \{0, 1\}) \setminus \{\{\}\}] \\ & & \land\; y \in [0 \,..\, (N-1) \to \{0, 1\}] \\ & & \land\; pc \in [0 \,..\, (N-1) \to \{\text{"a1"}, \text{"a2"}, \text{"b"}, \text{"Done"}\}] \end{array}$$

$$\begin{array}{lll} \mathit{Inv} & \triangleq & \land\; \mathit{TypeOK} \\ & & \land\; \forall\, i \in 0 \,..\, (N-1) : (pc[i] \in \{\text{"b"}, \text{"Done"}\}) \Rightarrow (x[i] = \{1\}) \\ & & \land\; \lor\; \exists\, i \in 0 \,..\, (N-1) : pc[i] \neq \text{"Done"} \\ & & \quad\; \lor\; \exists\, i \in 0 \,..\, (N-1) : y[i] = 1 \end{array}$$

The proof of invariance of *PCorrect* differs from the proof in module Simple only because the single action a has been replaced by the two actions $a1$ and $a2$, and because the proof that b maintains the truth of the invariant required one extra decomposition to allow $Z3$ to prove it. As before, the decomposition of the proof of $\langle 1\rangle 2$ was essentially generated with the *Toolbox*'s Decompose Proof command.

THEOREM $\mathit{Correctness} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{PCorrect}$

$\langle 1\rangle$ USE $\mathit{NAssump}$ DEF $\mathit{ProcSet}$

$\langle 1\rangle 1.$ $\mathit{Init} \Rightarrow \mathit{Inv}$

  $\langle 2\rangle 1.$ $\mathit{Init} \Rightarrow 0 \in 0 \,..\, (N-1) \land pc[0] \neq \text{"Done"}$

    BY DEF Init

  $\langle 2\rangle$.QED BY $\langle 2\rangle 1$ DEF $\mathit{Init}, \mathit{Inv}, \mathit{TypeOK}$

$\langle 1\rangle 2.$ $\mathit{Inv} \land [\mathit{Next}]_{vars} \Rightarrow \mathit{Inv}'$

  $\langle 2\rangle$ SUFFICES ASSUME Inv,
                    $[\mathit{Next}]_{vars}$
           PROVE Inv'

    OBVIOUS

  $\langle 2\rangle 1.$ ASSUME NEW $\mathit{self} \in 0 \,..\, N-1$,
           $a1(\mathit{self})$
      PROVE Inv'

    BY $\langle 2\rangle 1$ DEF $a1, \mathit{Inv}, \mathit{TypeOK}$

  $\langle 2\rangle 2.$ ASSUME NEW $\mathit{self} \in 0 \,..\, N-1$,
           $a2(\mathit{self})$
      PROVE Inv'

    BY $\langle 2\rangle 2$ DEF $a2, \mathit{Inv}, \mathit{TypeOK}$

  $\langle 2\rangle 3.$ ASSUME NEW $\mathit{self} \in 0 \,..\, N-1$,
           $b(\mathit{self})$
      PROVE Inv'

    $\langle 3\rangle$ SUFFICES ASSUME NEW $v \in x[(\mathit{self}-1)\%N]$,
                      $y' = [y \text{ EXCEPT } ![\mathit{self}] = v]$
             PROVE Inv'

      BY $\langle 2\rangle 3$ DEF b

    $\langle 3\rangle$ QED

```
        BY $\langle 2\rangle 3$, $Z3$ DEF $b$, $Inv$, $TypeOK$
  $\langle 2\rangle 4$.CASE UNCHANGED $vars$
    BY $\langle 2\rangle 4$  DEF $TypeOK$, $Inv$, $vars$
  $\langle 2\rangle 5$. QED
    BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 2\rangle 4$  DEF $Next$, $Terminating$, $proc$
$\langle 1\rangle 3$. $Inv \Rightarrow PCorrect$
  BY  DEF $Inv$, $TypeOK$, $PCorrect$
$\langle 1\rangle 4$. QED
  BY $\langle 1\rangle 1$, $\langle 1\rangle 2$, $\langle 1\rangle 3$, $PTL$ DEF $Spec$
```

As for the specification in module Simple, the proof can be shortened avoiding the decomposition of step ⟨1⟩2. In order to find the witness for the existential quantifier, the proof of step ⟨1⟩1 requires a small hint showing that 0 is a suitable value.

```
THEOREM $Correctness2 \triangleq Spec \Rightarrow \Box PCorrect$
$\langle 1\rangle$.USE $NAssump$ DEF $Inv$, $TypeOK$, $ProcSet$
$\langle 1\rangle 1$. $Init \Rightarrow Inv$
  BY $0 \in 0 \mathrel{..} N - 1$ DEF $Init$
$\langle 1\rangle 2$. $Inv \land [Next]_{vars} \Rightarrow Inv'$
  BY  DEF $Next$, $vars$, $a1$, $a2$, $b$, $Terminating$, $proc$
$\langle 1\rangle 3$. $Inv \Rightarrow PCorrect$
  BY  DEF $PCorrect$
$\langle 1\rangle$.QED  BY $\langle 1\rangle 1$, $\langle 1\rangle 2$, $\langle 1\rangle 3$, $PTL$ DEF $Spec$
```

\* Modification History
\* Last modified *Tue* May 14 07:18:15 *PDT* 2019 by *lamport*
\* Created *Mon Apr* 15 16:25:14 *PDT* 2019 by *lamport*