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 EATS (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) |-> {0}], y = [i \in 0..(N-1) |-> 0] ;
    process ( proc \in 0..N-1 ) {
      a1: x[self] := {0, 1} ;
      a2: x[self] := {1} ;
```

b: **with** ($v \in x[(self - 1)\%N]$) *f* $y[self] := v$ *g*
g
g

****

BEGIN TRANSLATION

VARIABLES $x,\ y,\ pc$

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

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

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

$$a1(self) \triangleq \begin{aligned}[t]&\land pc[self] = \text{"a1"}\\ &\land x' = [x \text{ EXCEPT } ![self] = \{0, 1\}]\\ &\land pc' = [pc \text{ EXCEPT } ![self] = \text{"a2"}]\\ &\land y' = y\end{aligned}$$

$$a2(self) \triangleq \begin{aligned}[t]&\land pc[self] = \text{"a2"}\\ &\land x' = [x \text{ EXCEPT } ![self] = \{1\}]\\ &\land pc' = [pc \text{ EXCEPT } ![self] = \text{"b"}]\\ &\land y' = y\end{aligned}$$

$$b(self) \triangleq \begin{aligned}[t]&\land pc[self] = \text{"b"}\\ &\land \exists v \in x[(self-1)\%N] :\\ &\qquad y' = [y \text{ EXCEPT } ![self] = v]\\ &\land pc' = [pc \text{ EXCEPT } ![self] = \text{"Done"}]\\ &\land x' = x\end{aligned}$$

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

$Next \triangleq (\exists\, self \in 0\,..\,N-1 : proc(self))$
$\qquad \lor$ Disjunct to prevent deadlock on termination
$\qquad\quad ((\forall\, self \in ProcSet : pc[self] = \text{"Done"}) \land \text{UNCHANGED } vars)$

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

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

END TRANSLATION

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

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

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*.

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

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

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 ⟨1⟩2 was essentially generated with the *Toolbox*'s Decompose Proof command.

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

⟨1⟩ USE $\mathit{NAssump}$

⟨1⟩1. $\mathit{Init} \Rightarrow \mathit{Inv}$

  BY DEF $\mathit{Init}, \mathit{Inv}, \mathit{TypeOK}, \mathit{ProcSet}$

⟨1⟩2. $\mathit{Inv} \land [\mathit{Next}]_{\mathsf{vars}} \Rightarrow \mathit{Inv}'$

  ⟨2⟩ SUFFICES ASSUME Inv,
              $[\mathit{Next}]_{\mathsf{vars}}$
        PROVE Inv'

    OBVIOUS

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

    BY ⟨2⟩1 DEF $a1, \mathit{Inv}, \mathit{TypeOK}$

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

    BY ⟨2⟩2 DEF $a2, \mathit{Inv}, \mathit{TypeOK}$

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

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

      BY ⟨2⟩3 DEF b

    ⟨3⟩ QED

      BY ⟨2⟩3, $Z3$ DEF $b, \mathit{Inv}, \mathit{TypeOK}$

  ⟨2⟩4. CASE UNCHANGED vars

    BY ⟨2⟩4 DEF $\mathit{TypeOK}, \mathit{Inv}, \mathit{vars}$

  ⟨2⟩5. QED

BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 2\rangle 4$ DEF *Next*, *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*

* 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*