MODULE *Counter*

TLA+ module for Op-based *Counter*. See its implementation in paper *Burckhardt@POPL'*2014.
We check that the Op-based *Counter* satisfies the strong eventual convergence property (*SEC*)

EXTENDS *Naturals*, *Sequences*, *Bags*, *TLC*

CONSTANTS

Protocol variables.

Replica, the set of replicas

Auxiliary variables for model checking.

Max *Max*[*r*]: the maximum number of the *Inc*() event replica $r \in Replica$ can issue; for finite-state model checking

VARIABLES

Protocol variables.

counter, *counter*[*r*]: the current value of the counter at replica $r \in Replica$
acc, *acc*[*r*]: the number of increments performed since the last broadcast at replica $r \in Replica$
incoming, *incoming*[*r*]: incoming messages at replica $r \in Replica$

Auxiliary variables for model checking.

inc *inc*[*r*]: the number of *Inc*() events issued by the replica $r \in Replica$; for finite-state model checking

The type correctness predicate.

$$\begin{aligned} TypeOK \triangleq\ & \land counter \in [Replica \to Nat] \\ & \land acc \in [Replica \to Nat] \\ & \land incoming \in [Replica \to SubBag(SetToBag(Nat))] \ \backslash * \text{ message ordering is not important; using bag (}i.e.\text{, mul} \\ & \land inc \in [Replica \to Nat] \end{aligned}$$

The initial state predicate.

$$\begin{aligned} Init \triangleq\ & \land counter = [r \in Replica \mapsto 0] \\ & \land acc = [r \in Replica \mapsto 0] \\ & \land incoming = [r \in Replica \mapsto EmptyBag] \\ & \land inc = [r \in Replica \mapsto 0] \end{aligned}$$

Replica $r \in Replica$ issues an *Inc*() event.

$$\begin{aligned} Inc(r) \triangleq\ & \land \text{TRUE} && \text{no pre-condition} \\ & \land counter' = [counter \text{ EXCEPT } ![r] = @ + 1] && \text{current } counter + 1 \\ & \land acc' = [acc \text{ EXCEPT } ![r] = @ + 1] && \#\text{ of accumulated increments } + 1 \\ & \land inc' = [inc \text{ EXCEPT } ![r] = @ + 1] && \#\text{ of increments } + 1 \\ & \land \text{UNCHANGED } \langle incoming \rangle \end{aligned}$$

Broadcast a message m to all replicas except the sender s.

$$\begin{aligned} Broadcast(s, m) \triangleq\ & [r \in Replica \mapsto \\ & \quad \text{IF } s = r \\ & \quad \text{THEN } incoming[s] \end{aligned}$$

```
67                                  ELSE  incoming[r] ⊕ SetToBag({m})]

    Replica r issues a Send() event, sending an update message.
72  Send(r) ≜  ∧ acc[r] ≠ 0        there are accumulated increments
73             ∧ acc' = [acc EXCEPT ![r] = 0]    reset acc[r]
74             ∧ incoming' = Broadcast(r, acc[r])    broadcast acc[r] to other replicas
75             ∧ UNCHANGED ⟨counter, inc⟩

    Replica r issues a Receive() event, receiving an update message.
80  Receive(r) ≜  ∧ incoming[r] ≠ EmptyBag    there are accumulated increments from other replicas
81                ∧ ∃ m ∈ BagToSet(incoming[r]) :    message reordering can be tolerant
82                    ( ∧ counter' = [counter EXCEPT ![r] = @ + m]
83                      ∧ incoming' = [incoming EXCEPT ![r] = @ ⊖ SetToBag({m})])    each message is delivered exa
84                ∧ UNCHANGED ⟨acc, inc⟩

86  ──────────────────────────────────────────────────────────────────────

    The Next-state relation.
90  Next ≜  ∧ ∃ r ∈ Replica : Inc(r) ∨ Send(r) ∨ Receive(r)

    The specification.
95  vars ≜ ⟨counter, acc, incoming, inc⟩
96  Spec ≜ Init ∧ □[Next]_vars ∧ WF_vars(Next)

98  ──────────────────────────────────────────────────────────────────────

    A state constraint that is useful for validating the specification using finite-state model checking:
    each replica r ∈ Replica can issue at most Max[r]Inc() events.
104 IncConstraint ≜ ∀ r ∈ Replica : inc[r] ≤ Max[r]

106 ──────────────────────────────────────────────────────────────────────

    The correctness of counter: Eventual Convergence (EC), Quiescent Consistency (QC), and Strong
    Eventual Convergence (SEC).

    Eventual Consistency (EC)
117 Convergence ≜ ∀ r, s ∈ Replica : (counter[r] = counter[s] ∧ counter[r] ≠ 0)    counter[r] ≠ 0: excluding the initial s
118 EC ≜ ◇Convergence

    Quiescent Consistency (QC)
123 AccBroadcast ≜ ∀ r ∈ Replica : acc[r] = 0    all accumulated increments have been broadcast
124 MessageDelivery ≜ ∀ r ∈ Replica : incoming[r] = EmptyBag    all messages have been delivered
125 QConvergence ≜ ∀ r, s ∈ Replica : counter[r] = counter[s]    no counter[r] ≠ 0

127 QC ≜ □((AccBroadcast ∧ MessageDelivery) ⇒ QConvergence)

    Strong Eventual Consistency (SEC)
132 ──────────────────────────────────────────────────────────────────────
```

\ * Modification History
\ * Last modified *Thu Aug* 02 15:02:12 *CST* 2018 by *hengxin*
\ * Created Sun *Jun* 03 20:08:57 *CST* 2018 by *hengxin*