```
1  ---------- MODULE GryadkaCasRegister ----------
   (* Written by Greg Rogers
      TLA+ code: https://gist.github.com/grogers0/c7e87f9dfe58c6070b19db9d3c073b72
      Post (A TLA+ specification for Gryadka): https://medium.com/@grogepodge/tla-specification-for-gryadka-c80cd625944e *)
10 EXTENDS Integers, Sequences, FiniteSets
11 ----------
12 (* Timestamps is the set of possible timestamps for operations to choose from.
13    Each operation uses a unique timestamp.
14    Values is the set of possible values to set the register to.
15    Acceptors is the set of nodes which act as acceptors in the paxos sense.
16    Quorums is the set of all possible quorums, typically simple majorities. *)
17 CONSTANTS Timestamps, Values, Acceptors, Quorums

19 ASSUME Timestamps \subseteq Nat
20 ASSUME IsFiniteSet(Timestamps)
21 NoTS \triangleq -1
22 ASSUME NoTS \notin Timestamps

24 ASSUME Quorums \subseteq SUBSET Acceptors
25 ASSUME \forall q1, q2 \in Quorums : q1 \cap q2 \neq {}

27 (* The initial value is chosen arbitrarily *)
28 InitVal \triangleq CHOOSE v \in Values : TRUE

30 (* msgs is the buffer of all messages. Messages can be delivered out of order or duplicated.
31    ops is the mapping from timestamp to CAS(old, new) for operations being proposed.
32    acceptorTS is the timestamp each acceptor is prepared for, only operations which match this value are accepted.
33    acceptorValTS is the timestamp of the last accepted value for each acceptor, or NoTS is none has been accepted yet.
34    acceptorValue is the last accepted value for each acceptor, or InitVal if none has been accepted yet.
35    history is the actual order of invoke/response actions for the operations identified by the timestamp. *)
36 VARIABLES msgs, ops, acceptorTS, acceptorValTS, acceptorValue, history
37 ----------
38 Messages \triangleq [type : {"prepare"}, acceptor : Acceptors, ts : Timestamps]
39       \cup [type       : {"promise"}, acceptor : Acceptors, ts : Timestamps,
40             prevTS : Timestamps \cup {NoTS}, prevVal : Values]
41       \cup [type : {"accept"}, acceptor : Acceptors, ts : Timestamps, val : Values]
42       \cup [type : {"accepted"}, acceptor : Acceptors, ts : Timestamps, val : Values]
43 (* Each operation represents a CAS from an oldVal to a newVal. In Gryadka,
44    reads are treated the same as CAS(val, val) *)
45 Operations \triangleq [oldVal : Values, newVal : Values]
46 Events \triangleq [type : {"invoke", "response"}, ts : Timestamps]

48 TypeOK \triangleq \land msgs \subseteq Messages
49              \land ops \in [Timestamps \to Operations]
50              \land acceptorTS \in [Acceptors \to Timestamps \cup {NoTS}]
```

```
51              ∧ acceptorValTS ∈ [Acceptors → Timestamps ∪ {NoTS}]
52              ∧ acceptorValue  ∈ [Acceptors → Values]
53              ∧ history ∈ Seq(Events)

55  Init ≜  ∧ msgs = {}
56          ∧ ops ∈ [Timestamps → Operations]
57          ∧ acceptorTS = [a ∈ Acceptors ↦ NoTS]
58          ∧ acceptorValTS = [a ∈ Acceptors ↦ NoTS]
59          ∧ acceptorValue  = [a ∈ Acceptors ↦ InitVal]
60          ∧ history = ⟨⟩

62 ├──────────────────────────────────────────────────────────────────────────┤
63  SelectMessages(type, ts) ≜ {m ∈ msgs : m.type = type ∧ m.ts = ts}
64  PromisedValue(ts) ≜ LET promiseMsgs ≜ SelectMessages("promise", ts)
65                      IN  (CHOOSE m ∈ promiseMsgs : ∀ m2 ∈ promiseMsgs : m.prevTS ≥ m2.prevTS).prev
66 ├──────────────────────────────────────────────────────────────────────────┤
67  Prepare(ts) ≜  ∧ SelectMessages("prepare", ts) = {}  Each timestamp must be unique
68                 ∧ msgs′ = msgs ∪ [type : {"prepare"}, acceptor : Acceptors, ts : {ts}]
69                 ∧ history′ = Append(history, [type ↦ "invoke", ts ↦ ts])
70                 ∧ UNCHANGED ⟨ops, acceptorTS, acceptorValTS, acceptorValue⟩

72  RecvPrepare(a, ts) ≜  ∧ acceptorTS[a] = NoTS ∨ acceptorTS[a] < ts
73                        ∧ acceptorTS′ = [acceptorTS EXCEPT ![a] = ts]
74                        ∧ msgs′ = msgs ∪ {[type ↦ "promise", acceptor ↦ a, ts ↦ ts,
75                                          prevTS ↦ acceptorValTS[a], prevVal ↦ acceptorValue[a]]}
76                        ∧ UNCHANGED ⟨ops, acceptorValTS, acceptorValue, history⟩

78  Accept(ts) ≜  ∧ {m.acceptor : m ∈ SelectMessages("promise", ts)} ∈ Quorums
79                ∧ ops[ts].oldVal = PromisedValue(ts)
80                ∧ msgs′ = msgs ∪ [type : {"accept"}, acceptor : Acceptors, ts : {ts}, val : {ops[ts].newVal}]
81                ∧ UNCHANGED ⟨ops, acceptorTS, acceptorValTS, acceptorValue, history⟩

83  RecvAccept(a, ts, v) ≜  ∧ acceptorTS[a] = ts
84                          ∧ acceptorValTS′ = [acceptorValTS EXCEPT ![a] = ts]
85                          ∧ acceptorValue′  = [acceptorValue EXCEPT ![a] = v]
86                          ∧ msgs′ = msgs ∪ [type : {"accepted"}, acceptor : {a}, ts : {ts}, val : {v}]
87                          ∧ UNCHANGED ⟨ops, acceptorTS, history⟩

89  Accepted(ts) ≜  ∧ {m.acceptor : m ∈ SelectMessages("accepted", ts)} ∈ Quorums
90                  ∧ {hpos ∈ DOMAIN history : history[hpos] = [type ↦ "response", ts ↦ ts]} = {}
91                  ∧ history′ = Append(history, [type ↦ "response", ts ↦ ts])
92                  ∧ UNCHANGED ⟨msgs, ops, acceptorTS, acceptorValTS, acceptorValue⟩

94  Next ≜  ∨ ∃ ts ∈ Timestamps :  ∨ Prepare(ts)
95                                 ∨ Accept(ts)
96                                 ∨ Accepted(ts)
97          ∨ ∃ m ∈ msgs :  ∨ m.type = "prepare" ∧ RecvPrepare(m.acceptor, m.ts)
```

```
                                    ∨ m.type = "accept" ∧ RecvAccept(m.acceptor, m.ts, m.val)

Spec ≜ Init ∧ □[Next]⟨msgs, ops, acceptorTS, acceptorValTS, acceptorValue, history⟩
```

```
FiniteSeq(S) ≜ UNION {[1 .. n → S] : n ∈ 1 .. Cardinality(S)}
SeqAsSet(S) ≜ {S[i] : i ∈ DOMAIN S}

HistoryIsLinearizable ≜ ∃ order ∈ {⟨⟩} ∪ FiniteSeq(Timestamps) :
    ∧ ∀ H ∈ SeqAsSet(history) : H.type = "response" ⇒ H.ts ∈ SeqAsSet(order)
    ∧ ∀ H1_i, H2_i ∈ DOMAIN history :
        (history[H1_i].type = "response" ∧ history[H2_i].type = "invoke" ∧ H1_i < H2_i) ⇒
            (
                history[H2_i].ts ∈ SeqAsSet(order) ⇒
                    ∃ i1, i2 ∈ DOMAIN order :
                        ∧ order[i1] = history[H1_i].ts
                        ∧ order[i2] = history[H2_i].ts
                        ∧ i1 < i2
            )
    ∧ ∀ i1, i2 ∈ DOMAIN order :
        i2 = i1 + 1 ⇒ ops[order[i1]].newVal = ops[order[i2]].oldVal
    ∧ order ≠ ⟨⟩ ⇒ InitVal = ops[order[1]].oldVal

Inv ≜ ∧ TypeOK
      ∧ HistoryIsLinearizable
```
