```
1  ---------- MODULE GryadkaCasRegister ----------
2    Written by Greg Rogers
3    TLA+ code: https://gist.github.com/grogers0/c7e87f9dfe58c6070b19db9d3c073b72
4    Post (A TLA+ specification for Gryadka): https://medium.com/@grogepodge/tla-specification-for-gryadka-c80cd625944e
5  ----------
6  EXTENDS Integers, Sequences, FiniteSets
7  ----------
8    Timestamps is the set of possible timestamps for operations to choose from.
9      Each operation uses a unique timestamp.
10   Values is the set of possible values to set the register to.
11   Acceptors is the set of nodes which act as acceptors in the paxos sense.
12   Quorums is the set of all possible quorums, typically simple majorities.
13 CONSTANTS Timestamps, Values, Acceptors, Quorums

15 ASSUME Timestamps ⊆ Nat
16 ASSUME IsFiniteSet(Timestamps)
17 NoTS ≜ −1
18 ASSUME NoTS ∉ Timestamps

20 ASSUME Quorums ⊆ SUBSET Acceptors
21 ASSUME ∀ q1, q2 ∈ Quorums : q1 ∩ q2 ≠ {}

23   The initial value is chosen arbitrarily
24 InitVal ≜ CHOOSE v ∈ Values : TRUE

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

44 TypeOK ≜ ∧ msgs ⊆ Messages
45          ∧ ops ∈ [Timestamps → Operations]
46          ∧ acceptorTS ∈ [Acceptors → Timestamps ∪ {NoTS}]
```

```
47              ∧ acceptorValTS ∈ [Acceptors → Timestamps ∪ {NoTS}]
48              ∧ acceptorValue  ∈ [Acceptors → Values]
49              ∧ history ∈ Seq(Events)

51  Init ≜  ∧ msgs = {}
52          ∧ ops ∈ [Timestamps → Operations]
53          ∧ acceptorTS = [a ∈ Acceptors ↦ NoTS]
54          ∧ acceptorValTS = [a ∈ Acceptors ↦ NoTS]
55          ∧ acceptorValue  = [a ∈ Acceptors ↦ InitVal]
56          ∧ history = ⟨⟩

58 ├──────────────────────────────────────────────────────────────────────────┤
59  SelectMessages(type, ts) ≜ {m ∈ msgs : m.type = type ∧ m.ts = ts}
60  PromisedValue(ts) ≜ LET promiseMsgs ≜ SelectMessages("promise", ts)
61                      IN  (CHOOSE m ∈ promiseMsgs : ∀ m2 ∈ promiseMsgs : m.prevTS ≥ m2.prevTS).prev
62 ├──────────────────────────────────────────────────────────────────────────┤
63  Prepare(ts) ≜  ∧ SelectMessages("prepare", ts) = {}  Each timestamp must be unique
64                 ∧ msgs′ = msgs ∪ [type : {"prepare"}, acceptor : Acceptors, ts : {ts}]
65                 ∧ history′ = Append(history, [type ↦ "invoke", ts ↦ ts])
66                 ∧ UNCHANGED ⟨ops, acceptorTS, acceptorValTS, acceptorValue⟩

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

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

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

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

90  Next ≜  ∨ ∃ ts ∈ Timestamps :  ∨ Prepare(ts)
91                                 ∨ Accept(ts)
92                                 ∨ Accepted(ts)
93          ∨ ∃ 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

└──────────────────────────────────────────────────────────────────────────────
```