─────────────────── MODULE *MapCache* ───────────────────

EXTENDS *Naturals*, *FiniteSets*, *Sequences*, *TLC*

An empty value
CONSTANT *Nil*

The set of possible keys
CONSTANT *Key*

The set of possible values
CONSTANT *Value*

The system state
VARIABLE *state*

The cache state
VARIABLE *cache*

The current cache version
VARIABLE *cacheVersion*

The highest version read by the client
VARIABLE *readVersion*

A sequence of update events
VARIABLE *events*

The maximum version assigned to an event
VARIABLE *version*

The history of reads for the client; used by the model checker to verify sequential consistency
VARIABLE *reads*

$vars \triangleq \langle state,\ cache,\ cacheVersion,\ readVersion,\ events,\ version,\ reads \rangle$

───

The type invariant checks that the client's reads never go back in time
$TypeInvariant \triangleq$
    $\wedge\ \forall\, k \in Key :$
        $\wedge\ \forall\, r \in \text{DOMAIN } reads[k] :$
            $r > 1 \Rightarrow reads[k][r] \geq reads[k][r-1]$

───

This section models helpers for managing the system and cache state

Drop a key from the domain of a function
$DropKey(s,\ k) \triangleq [i \in \text{DOMAIN } s \setminus \{k\} \mapsto s[i]]$

1

$PutEntry(s,\ e)\ \triangleq$
    IF $e.key \in$ DOMAIN s THEN
        $[s$ EXCEPT $![e.key] = e]$
    ELSE
        $s \,@@\,(e.key :> e)$

This section models the method calls for the Map primitive. Map entries can be created, updated, deleted, and read. When the map state is changed, events are enqueued for the client, and the learner updates the cache.

Get a value/version for a key in the map
$Get(k)\ \triangleq$
    $\wedge\ \vee\ \wedge\ cacheVersion \geq readVersion$
        $\wedge\ k \in$ DOMAIN $cache$
        $\wedge\ reads' = [reads$ EXCEPT $![k] = Append(reads[k],\ cache[k].version)]$
        $\wedge$ UNCHANGED $\langle readVersion \rangle$
      $\vee\ \wedge\ k \notin$ DOMAIN $cache$
        $\wedge\ k \in$ DOMAIN $state$
        $\wedge\ reads' = [reads$ EXCEPT $![k] = Append(reads[k],\ state[k].version)]$
        $\wedge\ readVersion' = state[k].version$
      $\vee\ \wedge\ k \notin$ DOMAIN $cache$
        $\wedge\ k \notin$ DOMAIN $state$
        $\wedge\ reads' = [reads$ EXCEPT $![k] = Append(reads[k],\ version)]$
        $\wedge\ readVersion' = version$
    $\wedge$ UNCHANGED $\langle state,\ cache,\ cacheVersion,\ events,\ version \rangle$

Put a key/value pair in the map
$Put(k,\ v)\ \triangleq$
    $\wedge\ version' = version + 1$
    $\wedge$ LET $entry\ \triangleq\ [key \mapsto k,\ value \mapsto v,\ version \mapsto version']$
      IN
        $\wedge\ state' = PutEntry(state,\ entry)$
        $\wedge\ events' = Append(events,\ entry)$
        $\wedge\ cache' = DropKey(cache,\ k)$
    $\wedge$ UNCHANGED $\langle cacheVersion,\ readVersion,\ reads \rangle$

Remove a key from the map
$Remove(k)\ \triangleq$
    $\wedge\ k \in$ DOMAIN $state$
    $\wedge\ version' = version + 1$
    $\wedge$ LET $entry\ \triangleq\ [key \mapsto k,\ value \mapsto Nil,\ version \mapsto version']$
      IN
        $\wedge\ state' = DropKey(state,\ k)$
        $\wedge\ events' = Append(events,\ entry)$

$$\wedge\ cache' = DropKey(cache,\ k)$$
$$\wedge\ \textsc{unchanged}\ \langle cacheVersion,\ readVersion,\ reads\rangle$$

Learn of a map update

$Learn\ \triangleq$
$\quad \wedge\ Cardinality(\textsc{domain}\ events) > 0$
$\quad \wedge\ \textsc{let}\ entry\ \triangleq\ events[1]$
$\qquad \textsc{in}$
$\qquad\quad \wedge\ \vee\ \wedge\ entry.value \neq Nil$
$\qquad\qquad\qquad \wedge\ cache' = PutEntry(cache,\ entry)$
$\qquad\qquad\ \vee\ \wedge\ entry.value = Nil$
$\qquad\qquad\qquad \wedge\ cache' = DropKey(cache,\ entry.key)$
$\qquad\quad \wedge\ cacheVersion' = entry.version$
$\quad \wedge\ events' = SubSeq(events,\ 2,\ Len(events))$
$\quad \wedge\ \textsc{unchanged}\ \langle state,\ version,\ readVersion,\ reads\rangle$

$Init\ \triangleq$
$\quad \wedge\ state\ = [i \in \{\} \mapsto [key \mapsto Nil,\ value \mapsto Nil,\ version \mapsto Nil]]$
$\quad \wedge\ cache = [i \in \{\} \mapsto [key \mapsto Nil,\ value \mapsto Nil,\ version \mapsto Nil]]$
$\quad \wedge\ events = [i \in \{\} \mapsto [key \mapsto Nil,\ value \mapsto Nil,\ version \mapsto Nil]]$
$\quad \wedge\ version = 0$
$\quad \wedge\ cacheVersion = 0$
$\quad \wedge\ readVersion = 0$
$\quad \wedge\ reads = [k \in Key \mapsto \langle\rangle]$

$Next\ \triangleq$
$\quad \vee\ \exists\,k \in Key : \exists\,v \in Value : Put(k,\ v)$
$\quad \vee\ \exists\,k \in Key : Get(k)$
$\quad \vee\ \exists\,k \in Key : Remove(k)$
$\quad \vee\ Learn$

$Spec\ \triangleq\ Init \wedge \square[Next]_{\langle vars\rangle}$

\ * Modification History
\ * Last modified *Tue Feb* 11 02:22:52 *PST* 2020 by *jordanhalterman*
\ * Created *Mon Feb* 10 23:01:48 *PST* 2020 by *jordanhalterman*

3