MODULE *MapCache*

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

An empty value
CONSTANT *Nil*

The set of clients
CONSTANT *Client*

The set of possible keys
CONSTANT *Key*

The set of possible values
CONSTANT *Value*

The system state
VARIABLE *state*

The cache state
VARIABLE *cache*

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*

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

The type invariant checks that the client's reads never go back in time

$$
\begin{array}{l}
TypeInvariant \triangleq \\
\quad \land \forall\, c \in Client : \\
\qquad \land \forall\, k \in Key : \\
\qquad\quad \land \forall\, r \in \text{DOMAIN } reads[c][k] : \\
\qquad\qquad r > 1 \Rightarrow reads[c][k][r] \geq reads[c][k][r-1]
\end{array}
$$

This section models helpers for managing the system and cache state

Drop a key from the domain of a function

$DropKey(s,\, k) \triangleq [i \in \text{DOMAIN } s \setminus \{k\} \mapsto s[i]]$

Put an entry in the given function

$PutEntry(s,\, e) \triangleq$

```
    IF e.key ∈ 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(c, k) ≜
    ∧ ∨ ∧ k ∈ DOMAIN cache[c]
        ∧ reads' = [reads EXCEPT ![c][k] = Append(reads[c][k], cache[c][k].version)]
      ∨ ∧ k ∉ DOMAIN cache[c]
        ∧ k ∈ DOMAIN state
        ∧ reads' = [reads EXCEPT ![c][k] = Append(reads[c][k], state[k].version)]
      ∨ ∧ k ∉ DOMAIN cache[c]
        ∧ k ∉ DOMAIN state
        ∧ reads' = [reads EXCEPT ![c][k] = Append(reads[c][k], version)]
    ∧ UNCHANGED ⟨state, cache, events, version⟩
```

Put a key/value pair in the map

```
Put(c, k, v) ≜
    ∧ version' = version + 1
    ∧ LET entry ≜ [key ↦ k, value ↦ v, version ↦ version']
      IN
        ∧ state' = PutEntry(state, entry)
        ∧ events' = [i ∈ Client ↦ Append(events[i], entry)]
        ∧ cache' = [cache EXCEPT ![c] = PutEntry(cache[c], entry)]
    ∧ UNCHANGED ⟨reads⟩
```

Remove a key from the map

```
Remove(c, k) ≜
    ∧ k ∈ DOMAIN state
    ∧ version' = version + 1
    ∧ LET entry ≜ [key ↦ k, value ↦ Nil, version ↦ version']
      IN
        ∧ state' = DropKey(state, k)
        ∧ events' = [i ∈ Client ↦ Append(events[i], entry)]
        ∧ cache' = [cache EXCEPT ![c] = DropKey(cache[c], k)]
    ∧ UNCHANGED ⟨reads⟩
```

Learn of a map update

```
Learn(c) ≜
    ∧ Cardinality(DOMAIN events[c]) > 0
    ∧ LET entry ≜ events[c][1]
```

```
    IN
        ∨ ∧ entry.key ∈ DOMAIN cache[c]
          ∧ entry.version > cache[c][entry.key].version
          ∧ ∨ ∧ entry.value ≠ Nil
                ∧ cache' = [cache EXCEPT ![c] = PutEntry(cache[c], entry)]
            ∨ ∧ entry.value = Nil
                ∧ cache' = [cache EXCEPT ![c] = DropKey(cache[c], entry.key)]
        ∨ ∧ ∨ entry.key ∉ DOMAIN cache[c]
              ∨ ∧ entry.key ∈ DOMAIN cache[c]
                ∧ entry.version ≤ cache[c][entry.key].version
          ∧ UNCHANGED ⟨cache⟩
    ∧ events' = [events EXCEPT ![c] = SubSeq(events[c], 2, Len(events[c]))]
    ∧ UNCHANGED ⟨state, version, reads⟩

Evict a map entry from the cache
Evict(c, k) ≜
    ∧ k ∈ DOMAIN cache[c]
    ∧ cache' = [cache EXCEPT ![c] = DropKey(cache[c], k)]
    ∧ UNCHANGED ⟨state, events, version, reads⟩

----------

Init ≜
    ∧ state = [i ∈ {} ↦ [key ↦ Nil, value ↦ Nil, version ↦ Nil]]
    ∧ cache = [c ∈ Client ↦ [i ∈ {} ↦ [key ↦ Nil, value ↦ Nil, version ↦ Nil]]]
    ∧ events = [c ∈ Client ↦ [i ∈ {} ↦ [key ↦ Nil, value ↦ Nil, version ↦ Nil]]]
    ∧ version = 0
    ∧ reads = [c ∈ Client ↦ [k ∈ Key ↦ ⟨⟩]]

Next ≜
    ∨ ∃ c ∈ Client : ∃ k ∈ Key : ∃ v ∈ Value : Put(c, k, v)
    ∨ ∃ c ∈ Client : ∃ k ∈ Key : Get(c, k)
    ∨ ∃ c ∈ Client : ∃ k ∈ Key : Remove(c, k)
    ∨ ∃ c ∈ Client : Learn(c)
    ∨ ∃ c ∈ Client : ∃ k ∈ Key : Evict(c, k)

Spec ≜ Init ∧ □[Next]_⟨vars⟩

==========

\* Modification History
\* Last modified Tue Feb 11 09:50:05 PST 2020 by jordanhalterman
\* Created Mon Feb 10 23:01:48 PST 2020 by jordanhalterman
```