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

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

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) ≜
    IF e.key ∈ DOMAIN s THEN
        [s EXCEPT ![e.key] = e]
     ELSE
        s @@ (e.key :> e)

 Cache a map entry
Cache(entry) ≜ cache' = PutEntry(cache, entry)

 Evict a map key from the cache
Evict(k) ≜ cache' = DropKey(cache, k)
```

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) ≜
    ∧ ∨ ∧ cacheVersion ≥ readVersion
        ∧ k ∈ DOMAIN cache
        ∧ reads' = [reads EXCEPT ![k] = Append(reads[k], cache[k].version)]
        ∧ UNCHANGED ⟨readVersion⟩
      ∨ ∧ k ∉ DOMAIN cache
        ∧ k ∈ DOMAIN state
        ∧ reads' = [reads EXCEPT ![k] = Append(reads[k], state[k].version)]
        ∧ readVersion' = state[k].version
      ∨ ∧ k ∉ DOMAIN cache
        ∧ k ∉ DOMAIN state
        ∧ reads' = [reads EXCEPT ![k] = Append(reads[k], version)]
        ∧ readVersion' = version
    ∧ UNCHANGED ⟨state, cache, cacheVersion, events, version⟩

 Put a key/value pair in the map
Put(k, v) ≜
    ∧ version' = version + 1
    ∧ LET entry ≜ [key ↦ k, value ↦ v, version ↦ version'] 
      IN
        ∧ state' = PutEntry(state, entry)
        ∧ events' = Append(events, entry)
        ∧ Evict(k)
    ∧ UNCHANGED ⟨cacheVersion, readVersion, reads⟩

 Remove a key from the map
Remove(k) ≜
    ∧ version' = version + 1
```

$\land$ LET $entry \triangleq [key \mapsto k,\ version \mapsto version']$
IN
$\land\ state' = DropKey(state,\ k)$
$\land\ events' = Append(events,\ entry)$
$\land\ Evict(k)$
$\land$ UNCHANGED $\langle cacheVersion,\ readVersion,\ reads\rangle$

Learn of a map update

$Learn \triangleq$
$\land\ Cardinality(\text{DOMAIN } events) > 0$
$\land\ Cache(events[1])$
$\land\ cacheVersion' = events[1].version$
$\land\ events' = SubSeq(events,\ 2,\ Len(events))$
$\land$ UNCHANGED $\langle state,\ version,\ readVersion,\ reads\rangle$

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

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

$Spec \triangleq\ Init \land \Box[Next]_{\langle vars\rangle}$

---

* Modification History
* Last modified *Tue Feb* 11 01:49:12 *PST* 2020 by *jordanhalterman*
* Created *Mon Feb* 10 23:01:48 *PST* 2020 by *jordanhalterman*