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*

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*

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

---

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

---

This section models helpers for managing the system and cache state

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

Put an entry in the given function
$PutEntry(s, e) \triangleq$
$\quad \text{IF } e.key \in \text{DOMAIN } s \text{ THEN}$
$\quad\quad [s \text{ EXCEPT } ![e.key] = e]$
$\quad \text{ELSE}$
$\quad\quad s \mathbin{@@} (e.key :> e)$

Cache a map entry

$Cache(entry) \triangleq cache' = PutEntry(cache, entry)$

Evict a map key from the cache

$Evict(k) \triangleq cache' = DropKey(cache, k)$

---

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

$$
\begin{array}{l}
Get(k) \triangleq \\
\quad \land \lor \land k \in \text{DOMAIN } cache \\
\qquad \land reads' = Append(reads, cache[k].version) \\
\qquad \lor \land k \notin \text{DOMAIN } cache \\
\qquad \land k \in \text{DOMAIN } state \\
\qquad \land reads' = Append(reads, state[k].version) \\
\qquad \lor \land k \notin \text{DOMAIN } cache \\
\qquad \land k \notin \text{DOMAIN } state \\
\qquad \land reads' = Append(reads, version) \\
\quad \land \text{UNCHANGED } \langle state, cache, events, version \rangle
\end{array}
$$

Put a key/value pair in the map

$$
\begin{array}{l}
Put(k, v) \triangleq \\
\quad \land version' = version + 1 \\
\quad \land \text{LET } entry \triangleq [key \mapsto k, value \mapsto v, version \mapsto version'] \\
\quad \text{IN} \\
\qquad \land state' = PutEntry(state, entry) \\
\qquad \land events' = Append(events, entry) \\
\qquad \land Evict(k) \\
\quad \land \text{UNCHANGED } \langle reads \rangle
\end{array}
$$

Remove a key from the map

$$
\begin{array}{l}
Remove(k) \triangleq \\
\quad \land version' = version + 1 \\
\quad \land \text{LET } entry \triangleq [key \mapsto k, version \mapsto version'] \\
\quad \text{IN} \\
\qquad \land state' = DropKey(state, k) \\
\qquad \land events' = Append(events, entry) \\
\qquad \land Evict(k) \\
\quad \land \text{UNCHANGED } \langle reads \rangle
\end{array}
$$

Learn of a map update

$$
\begin{array}{l}
Learn \triangleq \\
\quad \land Cardinality(\text{DOMAIN } events) > 0 \\
\quad \land Cache(events[1])
\end{array}
$$

$\land\ events' = SubSeq(events,\ 2,\ Len(events))$
$\land$ UNCHANGED $\langle state,\ version,\ reads\rangle$

---

$Init\ \triangleq$
$\land\ state\ = [i \in \{\} \mapsto [key \mapsto Nil,\ value \mapsto Nil,\ version \mapsto Nil]]$
$\land\ cache = [i \in \{\} \mapsto [key \mapsto Nil,\ value \mapsto Nil,\ version \mapsto Nil]]$
$\land\ events = [i \in \{\} \mapsto [key \mapsto Nil,\ value \mapsto Nil,\ version \mapsto Nil]]$
$\land\ version = 0$
$\land\ reads = [i \in \{\} \mapsto 0]$

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

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

---

* Modification History
* Last modified *Tue Feb* 11 00:50:25 *PST* 2020 by *jordanhalterman*
* Created *Mon Feb* 10 23:01:48 *PST* 2020 by *jordanhalterman*