MODULE *naivecache*

EXTENDS *Naturals*

CONSTANTS
KEYS The full set of keys in the database

VARIABLES
database, *database*[*key*] = *DataVersion*
cache *cache*[*key*] = *CacheValue*

Imports cache requirements to test against
INSTANCE *cacherequirements*

$vars \triangleq \langle database, cache \rangle$

A cache can hold a hit or a miss for any given key
$CacheValue \triangleq CacheMiss \cup CacheHit$

$TypeOk \triangleq$
database is a mapping of keys to a data version
$\land\ database \in [KEYS \to DataVersion]$
cache is a mapping of kets to a cache value
$\land\ cache \in [KEYS \to CacheValue]$

$Init \triangleq$
All keys in the database are initialized to their first version
$\land\ database = [k \in KEYS \mapsto 0]$
All keys in the cache are initialized to a miss, *i.e.* no data in cache
$\land\ cache = [k \in KEYS \mapsto [type \mapsto \text{“miss”}]]$

$DatabaseUpdate(k) \triangleq$
The version of that key is incremented, representing a write
$\land\ database' = [database \text{ EXCEPT } ![k] = database[k] + 1]$
$\land$ UNCHANGED *cache*

$CacheRead(k) \triangleq$
The data is already in the cache
$\land\ cache[k] \in CacheHit$
So the cache remains the same
$\land$ UNCHANGED *cache*
$\land$ UNCHANGED *database*

$CacheReadThrough(k) \triangleq$
The data is not in the cache

$\land\ cache[k] \in CacheMiss$
So it is read from the database
$\land\ cache' = [cache$ EXCEPT
$!\,[k] = [$
Cache value is now a hit
$type \mapsto$ "hit",
Set to whatever version is in database
$version \mapsto database[k]$
$]$
$]$
$\land$ UNCHANGED $database$

$CacheEvict(k) \triangleq$
The data is in cache, so can be evicted
$\land\ cache[k] \in CacheHit$
$cache[k]$is turned into a miss
$\land\ cache' = [cache$ EXCEPT $!\,[k] = [type \mapsto$ "miss"$]]$
$\land$ UNCHANGED $database$

Fairness: Normally no operation is guaranteed to happen; it just may. However, that means that the cache could just stop reading forever. And so it would never update. Now that doesn't seem reasonable.

The cache will always be able to...
$CacheFairness \triangleq$
$\exists\, k \in KEYS :$
$\lor\ CacheRead(k)$ Read
$\lor\ CacheReadThrough(k)$ Write
$CacheEvict(k)$ is not here, because $CacheEvict$ is something that may happen. It is not guaranteed

Specification

$Next \triangleq$
$\exists\, k \in KEYS :$
Database states
$\lor\ DatabaseUpdate(k)$
Cache states
$\lor\ CacheRead(k)$
$\lor\ CacheReadThrough(k)$
$\lor\ CacheEvict(k)$

Cache fairness is included as part of the specification of system behavior.
This is just how the system works.
$Spec \triangleq\ Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(CacheFairness)$

\ * Modification History
\ * Last modified *Tue Jun* 14 22:51:06 *MST* 2022 by *elliotswart*
\ * Created *Tue Jun* 14 20:36:02 *MST* 2022 by *elliotswart*