Lazy Cache Implements Complete Cache
Draft Version[1]

P. B. Ladkin
Universität Bielefeld, Technische Fakultät
D-33501 Bielefeld, Germany
ladkin@techfak.uni-bielefeld.de

Sun 7 Jan 1996 [18:49]

[1]The author will be very grateful for notification of any errors!

Abstract

I provide a formal proof in TLA that the Lazy Cache algorithm of Afek, Brown and Merritt implements an abstraction called a Complete Cache. This proof is part of a verification in TLA that the Lazy Cache is sequentially consistent. I explain how the invariant was chosen, and comment the proof to allow it more easily to be read.

1 What's the Problem?

Rob Gerth set the challenge problem to verify formally that the Lazy Caching algorithm of Afek, Brown and Merritt [ABM93] is sequentially consistent [Ger96]. We undertook to verify the algorithm completely formally in TLA. The description of our proof appears in [LLRO96], which contains the TLA specifications and the approach on which the work described here is based. I present here part of the formal proof, namely the proof that the lazy caching algorithm implements an intermediate specification called the complete cache.

To find out how one goes about formulating specification and algorithm descriptions and verifying one against the other in TLA, see the report [Lam94]. I assume knowledge of TLA procedures.

2 How's It Solved?

The proof here, that the lazy cache implements the complete cache, is part of the entire proof that the lazy cache is sequentially consistent. The lazy caching algorithm works roughly as follows. There is a state machine corresponding to the picture in Fig. 3. Each processor writes a $\langle \mathit{datum}, \mathit{address} \rangle$ pair to an *out* queue (action *Write*). The *Head*s of these queues are non-deterministically selected for writing simultaneously and atomically to main memory and to all of the *in* queues, $in[i]$ for $i \in Proc$ (action *MemWrite*). The *in* queues hold the pending updates for the caches $c[i]$. A write from processor i is specially designated in $in[i]$, the cache-write queue for processor i itself. The caches $c[i]$ are updated from the queues $in[i]$ by a *CacheUpdate*, which takes the *Head* of queue $in[i]$ and writes it into the cache. There is no requirement that the caches remain coherent. Caches may lose values (via a *CacheInval*) operation) and may refresh (through *MemRead*). The 'trick' is in the processor read requests. Processor i may read from its cache $c[i]$ only when its out queue $out[i]$ is empty and there are no writes from processor i itself pending on its *in[i]* queue. This ensures that reads and writes from processor i are sequentially ordered in time, but yet there is no guarantee of time-like interleaving with other actions from other processors – a read from processor i may be executed before a timewise-previous write from processor j has been written from the queue $in[i]$ into i's cache $c[i]$. Sequential consistency and not time-like interleaving is the goal. Figures 1 and 2 contain the TLA specification of *LazyCache*.

For a part of the proof that Lazy Cache is sequentially consistent, we constructed an intermediate machine, called the Complete Cache, or *CCache*, pictured in Figure 4. *CCache* has no main memory, but each cache $cc[i]$ itself contains entries for the entire memory address space. Thus the memory is "virtual" in that it may be constructed out of any $cc[i]$, together with the updates contained in $in[i]$. Thus we may eliminate one variable (*mem*), and two actions (*MemRead* and *CacheInval*), absolving us from worrying about cache operations inessential for the proof of sequential consistency. The TLA specification for *CCache* appears in Figure 5.

In TLA, algorithms and specifications are both described by state machines. I must thus prove that the Lazy Cache machine implements the Complete Cache machine. Figure 6 states the correctness theorem to prove, along with correctness theorems for other parts of the proof. History variables are adequate to prove *LCimplCC*, which requires a non-trivial invariant for the safety part. (Proving sequential consistency from *CCache* requires prophecy variables also.) How this invariant arose is explained below. I don't claim that this is the "best" invariant, or the mathematically most slick. In fact, it seems rather messy. But this is engineering–everything should follow from method and rely less on mathematical insight. The invariant works, and it was developed from straightforward considerations as to how *LazyCache* must relate to *CCache*, so it suffices.

Various parts of the proof use the mathematics of *Sequences*, described in the module in Figure 7.

$$
\begin{array}{l}
\textbf{module } \mathit{LazyCache} \\
\textbf{import } \mathit{MemParams},\ \mathit{ChannelInterface},\ \mathit{Sequences},\ \mathit{Naturals} \\
\hline
\textbf{parameters } \ c, \mathit{in}, \mathit{out}, \mathit{mem} \ : \ \text{VARIABLE} \\
\hline
\end{array}
$$

$$\perp \ \triangleq \ \text{CHOOSE } i \ : \ i \notin \mathit{Data}$$

$$\mathit{Restrict}(f) \ \triangleq \ \{g \in [\mathit{Addr} \rightarrow \mathit{Data} \cup \{\perp\}] \ : \ \forall\, a \in \mathit{Addr} \ : \ g[a] \in \{f[a], \perp\}\}$$

$$
\begin{array}{lll}
\mathit{Init} & \triangleq & \land\ \mathit{mem} \in [\mathit{Addr} \rightarrow \mathit{InitData}] \\
 & & \land\ c \in [\mathit{Proc} \rightarrow \mathit{Restrict}(\mathit{mem})] \\
 & & \land\ \mathit{in} = [i \in \mathit{Proc} \mapsto \langle\,\rangle] \\
 & & \land\ \mathit{out} = [i \in \mathit{Proc} \mapsto \langle\,\rangle]
\end{array}
$$

$$
\begin{array}{lll}
\mathit{Read}(i, d, a) & \triangleq & \land\ \mathit{out}[i] = \langle\,\rangle \\
 & & \land\ \forall\, j \in 1\,..\,\mathit{Len}(\mathit{in}[i]) \ : \ \mathit{Len}(\mathit{in}[i][j]) = 2 \\
 & & \land\ c[i][a] = d \\
 & & \land\ \mathit{ChanOp}(\mathit{ch}[i], \langle\, i, \text{“Rd”}, d, a \,\rangle) \\
 & & \land\ \forall\, j \in \mathit{Proc} \setminus \{i\} \ : \ \text{UNCHANGED } \mathit{ch}[j] \\
 & & \land\ \text{UNCHANGED } \langle\, c, \mathit{in}, \mathit{out}, \mathit{mem} \,\rangle
\end{array}
$$

$$
\begin{array}{lll}
\mathit{Write}(i, d, a) & \triangleq & \land\ \mathit{ChanOp}(\mathit{ch}[i], \langle\, i, \text{“Wr”}, d, a \,\rangle) \\
 & & \land\ \mathit{out}' = [\mathit{out} \text{ EXCEPT } ![i] = \mathit{out}[i] \circ \langle\langle\, d, a \,\rangle\rangle] \\
 & & \land\ \forall\, j \in \mathit{Proc} \setminus \{i\} \ : \ \text{UNCHANGED } \mathit{ch}[j] \\
 & & \land\ \text{UNCHANGED } \langle\, c, \mathit{in}, \mathit{mem} \,\rangle
\end{array}
$$

$$
\begin{array}{l}
\mathit{MemWrite}(i) \ \triangleq \\
\quad \land\ \mathit{out}[i] \neq \langle\,\rangle \\
\quad \land\ \mathit{out}' = [\mathit{out} \text{ EXCEPT } ![i] = \mathit{Tail}(\mathit{out}[i])] \\
\quad \land\ \mathit{in}' = [j \in \mathit{Proc} \ \mapsto \ \mathit{in}[j] \circ \textbf{if } j = i \ \textbf{ then } \langle \mathit{Head}(\mathit{out}[i]) \circ \langle\, \text{“}*\text{”} \,\rangle \rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \ \textbf{else } \ \langle \mathit{Head}(\mathit{out}[i]) \rangle \qquad\qquad] \\
\quad \land\ \mathit{mem}' = [\mathit{mem} \text{ EXCEPT } ![\mathit{Head}(\mathit{out}[i])[2]] = \mathit{Head}(\mathit{out}[i])[1]\,] \\
\quad \land\ \text{UNCHANGED } \langle\, c, \mathit{ch} \,\rangle
\end{array}
$$

$$
\begin{array}{lll}
\mathit{CacheUpdate}(i) & \triangleq & \land\ \mathit{in}[i] \neq \langle\,\rangle \\
 & & \land\ \mathit{in}' = [\mathit{in} \text{ EXCEPT } ![i] = \mathit{Tail}(\mathit{in}[i])] \\
 & & \land\ c' = [c \text{ EXCEPT } ![i][\mathit{Head}(\mathit{in}[i])[2]] = \mathit{Head}(\mathit{in}[i])[1]\,] \\
 & & \land\ \text{UNCHANGED } \langle\, \mathit{out}, \mathit{ch}, \mathit{mem} \,\rangle
\end{array}
$$

Figure 1: The lazy caching algorithm (beginning).

Another module describes the operations *Chan* on the processor channels, but since these operations play the role of atomic formulae in the proof, I didn't include this module or its related modules here.

3 How's It All Work?

CCache turns cache refreshes (*MemRead*) and invalidations into things of the past. How may this happen? *MemRead*(i) pushes a cache refresh $\langle\, \mathit{data}, \mathit{address} \,\rangle$ pair on the end of the $\mathit{in}[i]$ queue. If this is to be a no-op, the new item must be invisible. *MemWrite* pushes a pair on the end of $\mathit{in}[i]$, too, but these pairs are not invisible. So 'visible' and 'invisible' writes must be distinguished. We

$$
\begin{array}{lcl}
\mathit{MemRead}(i) & \triangleq & \land\ \exists\, a \in \mathit{Addr} : \\
 & & \qquad \mathit{in}' = [\mathit{in}\ \text{EXCEPT}\ ![i] = \mathit{in}[i] \circ \langle \langle \mathit{mem}[a], a \rangle \rangle] \\
 & & \land\ \text{UNCHANGED}\ \langle \mathit{out}, c, \mathit{mem}, \mathit{ch} \rangle \\
\mathit{CacheInval}(i) & \triangleq & \land\ c' \in \{[c\ \text{EXCEPT}\ ![i] \in \mathit{Restrict}(c[i])]\} \\
 & & \land\ \text{UNCHANGED}\ \langle \mathit{in}, \mathit{out}, \mathit{mem}, \mathit{ch} \rangle \\
\mathit{Next}(i) & \triangleq & \lor\ \exists\, d \in \mathit{Data}, a \in \mathit{Addr} : \mathit{Read}(i, d, a) \lor \mathit{Write}(i, d, a) \\
 & & \lor\ \mathit{MemWrite}(i) \lor \mathit{CacheUpdate}(i) \lor \mathit{MemRead}(i) \\
 & & \quad \lor\ \mathit{CacheInval}(i) \\
\mathit{vars} & \triangleq & \langle c, \mathit{in}, \mathit{out}, \mathit{mem}, \mathit{ch} \rangle \\
\mathit{Spec} & \triangleq & \land\ \mathit{Init} \\
 & & \land\ \Box[\exists\, i \in \mathit{Proc} : \mathit{Next}(i)]_{\mathit{vars}} \\
 & & \land\ \forall\, i \in \mathit{Proc} : \mathrm{WF}_{\mathit{vars}}(\mathit{CacheUpdate}(i)) \land \mathrm{WF}_{\mathit{vars}}(\mathit{MemWrite}(i))
\end{array}
$$

Figure 2: The lazy caching algorithm (continued).

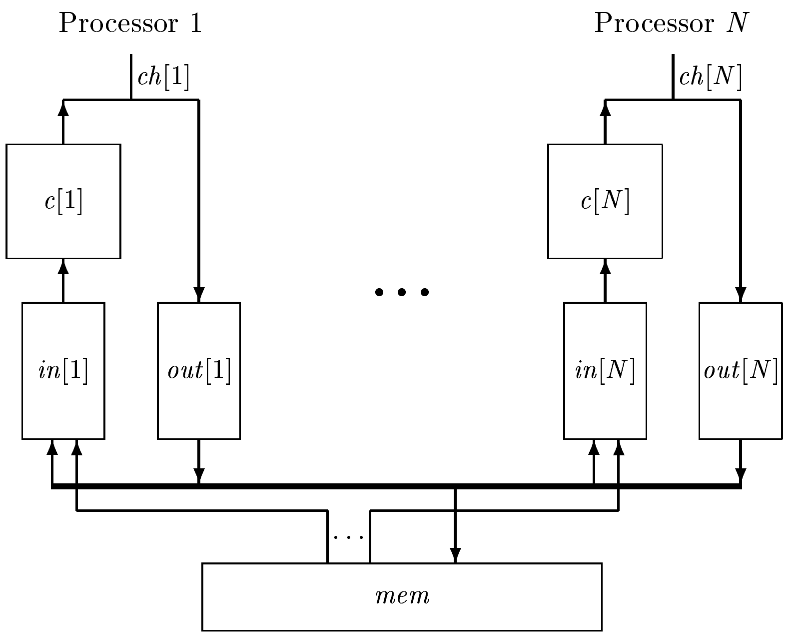

Figure 3: The state machine describing the lazy caching algorithm, where $c[i]$ is a cache, $\mathit{out}[i]$ is a queue of $\langle \mathit{data}, \mathit{address} \rangle$ pairs for writes by processor i that have not yet been performed to memory, and $\mathit{in}[i]$ is a queue of $\langle \mathit{data}, \mathit{address} \rangle$ pairs and $\langle \mathit{data}, \mathit{address}, \text{“}*\text{”} \rangle$ triples of pending writes to $c[i]$, a “$*$” indicating that the write was issued by processor i.

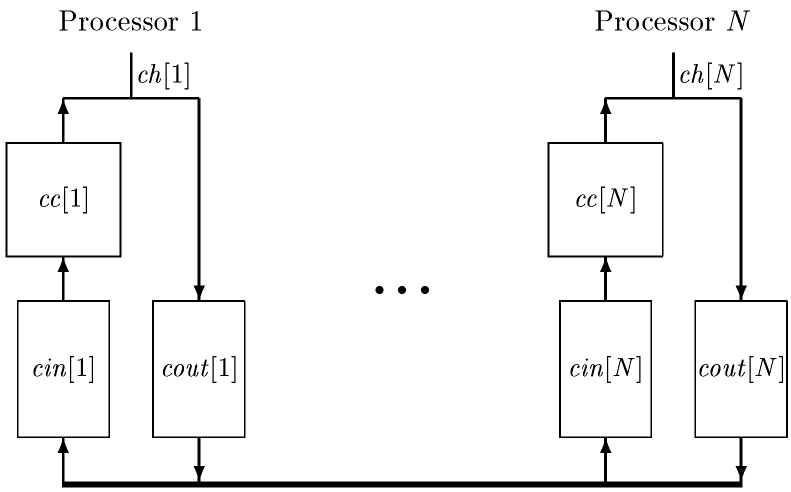

Figure 4: The state machine *CCache* describing the complete cache algorithm, where $cc[i]$ is a complete cache, and $cout[i]$ and $cin[i]$ are queues of $\langle processor, operation, data, address\rangle$ tuples.

simply add a field to the pair, telling us "who wrote me?" (Processor i if a refresh), and "how did I get here?" ("Rd" from a *MemRead*, "Wr" from a *MemWrite*). Thus the history variable *hin* has four-element entries instead of two-element entries. The effect of the operations *MemWrite* and *MemRead* on $in[i]$, and the virtual counterparts *HMemWrite* and *HMemRead* on $hin[i]$, is shown in Figures 8, 9, and 12.

How do we filter invisible writes from visible ones to form $cin[i]$? The *SelectSeq* function from *Sequences* is used with a predicate that distinguishes the two kinds of writes to filter the $hin[i]$ queue to form $cin[i]$. *SelectSeq* filters *hin* entries on the second element, "Rd" or "Wr".

When one of these to-be-filtered items (a "Rd") appears at the *Head* of $hin[i]$, the lazy-cache *CacheUpdate* that writes it into $c[i]$ must be a no-op for *CCache*. Thus $cc[i]$ must not change, hence it must contain the pair $\langle datum, address\rangle$ already. It certainly contains $\langle something, address\rangle$, since $cc[i]$ contains entries for all addresses. Thus, for this *CacheUpdate* of a refresh to be invisible, *something* had better already be *datum*. Figures 10 and 11 show the *CacheUpdate* and *HCacheUpdate* operations. What must **not** occur is shown in Figure 13.

Before constucting the invariant or proceeding with the machine simulation, I add the history variable *vc* for the complete cache, as well as other history variables. Hence, they may be explicitly mentioned in the invariant. The predicate *NotRefresh* identifies those 'real' items in an *in* queue, shown in Figure 14.

To rule out the 'bad' situation in Figure 13, I construct the predicate *FirstIsVC*. The predicate $FirstIsVC(a, i)$ says that all items in $in[i]$ with address a up to the first 'real' write are repeats of what's in the virtual cache $vc[i]$ already (*vc* will implement *cc*). *FirstIsVC* belongs to the invariant.

But that is not enough. The $vc[i]$ item $\langle d, a\rangle$ in Figure 13 got there somehow. It passed through the $hin[i]$ queue as a *NotRefresh* element. So, when there is a *NotRefresh* for address a in $hin[i]$, say in position m, then all refreshes of address a that follow it, up to the next *NotRefresh* of address a in $hin[i]$ in position n, must contain simply the repeated data value. This is expressed in the predicate $MemWritesFromTo(m, n, i)$, illustrated in Figure 15. The predicate $BetweenMemWrites(i)$ is simply the quantification of $MemWritesFromTo(m, n, i)$ over all m, n in $Len(in[i])$.

Figure 16 shows that the situation must also be avoided at the other end–all refreshes of address a following the last *NotRefresh* of a in $hin[i]$ must contain the same value as *mem*. If not, this

module *CCache*

import *MemParams, ChannelInterface, Sequences, Naturals*

parameters *cc, cin, cout* : VARIABLE

$$
\begin{array}{lll}
Init & \triangleq & \land\ cc \in [Proc \to [Addr \to InitData]] \\
 & & \land\ \forall\, i, j \in Proc\ :\ cc[i] = cc[j] \\
 & & \land\ cin = [i \in Proc \mapsto \langle\,\rangle] \\
 & & \land\ cout = [i \in Proc \mapsto \langle\,\rangle]
\end{array}
$$

$$
\begin{array}{lll}
Read(i, d, a) & \triangleq & \land\ cout[i] = \langle\,\rangle \\
 & & \land\ \forall\, j \in (1\,..\,Len(cin[i]))\ :\ cin[i][j][1] \neq i \\
 & & \land\ cc[i][a] = d \\
 & & \land\ ChanOp(ch[i], \langle i, \text{“Rd”}, d, a\rangle) \\
 & & \land\ \forall\, j \in Proc \setminus \{i\}\ :\ \textsc{unchanged}\ ch[j] \\
 & & \land\ \textsc{unchanged}\ \langle cc, cin, cout\rangle
\end{array}
$$

$$
\begin{array}{lll}
Write(i, d, a) & \triangleq & \land\ ChanOp(ch[i], \langle i, \text{“Wr”}, d, a\rangle) \\
 & & \land\ cout' = [cout\ \textsc{except}\ ![i] = cout[i] \circ \langle\langle i, \text{“Wr”}, d, a\rangle\rangle] \\
 & & \land\ \forall\, j \in Proc \setminus \{i\}\ :\ \textsc{unchanged}\ ch[j] \\
 & & \land\ \textsc{unchanged}\ \langle cc, cin\rangle
\end{array}
$$

$$
\begin{array}{lll}
MemWrite(i) & \triangleq & \land\ cout[i] \neq \langle\,\rangle \\
 & & \land\ cout' = [cout\ \textsc{except}\ ![i] = Tail(cout[i])] \\
 & & \land\ cin' = [j \in Proc \mapsto cin[j] \circ \langle Head(cout[i])\rangle] \\
 & & \land\ \textsc{unchanged}\ \langle cc, ch\rangle
\end{array}
$$

$$
\begin{array}{l}
CacheUpdate(i)\ \triangleq \\
\quad \land\ cin[i] \neq \langle\,\rangle \\
\quad \land\ cin' = [cin\ \textsc{except}\ ![i] = Tail(cin[i])] \\
\quad \land\ cc' = [cc\ \textsc{except}\ ![i][Head(cin[i])[4]] = Head(cin[i])[3]] \\
\quad \land\ \textsc{unchanged}\ \langle cout, ch\rangle
\end{array}
$$

$$
\begin{array}{lll}
Next(i) & \triangleq & \lor\ \exists\, d \in Data, a \in Addr\ :\ Read(i, d, a) \lor Write(i, d, a) \\
 & & \lor\ MemWrite(i) \lor CacheUpdate(i)
\end{array}
$$

$$
vars \triangleq \langle cc, cin, cout, ch\rangle
$$

$$
\begin{array}{lll}
Spec & \triangleq & \land\ Init \\
 & & \land\ \Box[\exists\, i \in Proc\ :\ Next(i)]_{vars} \\
 & & \land\ \forall\, i \in Proc\ :\ \mathrm{WF}_{vars}(CacheUpdate(i)) \land \mathrm{WF}_{vars}(MemWrite(i))
\end{array}
$$

Figure 5: The complete cache algorithm.

module *CacheCorrectness*

import *MemParams*

parameters c, in, out, mem : VARIABLE

---

include *LazyCache* **as** *LC*

$$
\begin{aligned}
&POp \triangleq [i \in Proc \mapsto \{i\} \times \{\text{“Rd”}, \text{“Wr”}\} \times Data \times Addr]\\
&OKOp(o, old, new) \triangleq \lor \land o[2] = \text{“Rd”}\\
&\qquad\qquad\qquad\qquad\quad\;\; \land o[3] = old[o[4]]\\
&\qquad\qquad\qquad\qquad\quad\;\; \land new = old\\
&\qquad\qquad\qquad\qquad\;\; \lor \land o[2] = \text{“Wr”}\\
&\qquad\qquad\qquad\qquad\quad\;\; \land new = [old \text{ EXCEPT } ![o[4]] = o[3]]\\
&InitDB \triangleq [Addr \to InitData]
\end{aligned}
$$

include *SeqDB1* **as** *DB1*

theorem $LCimpliesDB1 \triangleq LC.Spec \Rightarrow DB1.Spec$

---

include *CCache* **as** $CC(cc, cin, cout)$

theorems

$$
\begin{aligned}
&LCimpliesCC \triangleq LC.Spec \Rightarrow (\exists\, cc, cin, cout : CC(cc, cin, cout).Spec)\\
&CCimpliesDB1 \triangleq \forall\, cc, cin, cout : CC(cc, cin, cout).Spec \Rightarrow DB1.Spec
\end{aligned}
$$

---

include *ACCache* **as** $ACC(cc, cin, cout, vcq, vrq, vdch)$

theorems

$$
\begin{aligned}
&CCequivACC \triangleq\\
&\quad \forall\, cc, cin, cout : CC(cc, cin, cout).Spec \equiv\\
&\qquad (\exists\, vcq, vrq, vdch : ACC(cc, cin, cout, vcq, vrq, vdch).ASpec)\\
&ACCimpliesDB1 \triangleq\\
&\quad \forall\, cc, cin, cout, vcq, vrq, vdch :\\
&\qquad ACC(cc, cin, cout, vcq, vrq, vdch).ASpec \Rightarrow DB1.Spec
\end{aligned}
$$

Figure 6: The theorems constituting the correctness proof.

module *Sequences*

import *Naturals*

$$
\begin{array}{lcl}
m \,..\, n & \triangleq & \{i \in \mathit{Nat} : (m \leq i) \land (i \leq n)\} \\
\mathit{Len}(s) & \triangleq & \textsc{choose}\ n : (n \in \mathit{Nat}) \land ((\textsc{Domain}\ s) = (1 \,..\, n)) \\
\mathit{Head}(s) & \triangleq & s[1] \\
\mathit{Tail}(s) & \triangleq & [i \in 1 \,..(\mathit{Len}(s) - 1) \mapsto s[i+1]] \\
s \circ t & \triangleq & [i \in 1 \,..(\mathit{Len}(s) + \mathit{Len}(t)) \mapsto \mathbf{if}\ i \leq \mathit{Len}(s)\ \mathbf{then}\ s[i] \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \ \mathbf{else}\ \ t[i - \mathit{Len}(s)]] \\
\mathit{Seq}(S) & \triangleq & \textsc{union}\ \{[(1 \,..\, n) \rightarrow S] : n \in \mathit{Nat}\}
\end{array}
$$

$$
\begin{array}{l}
\mathit{SelectSeq}(s, \mathit{test}(\_)) \ \triangleq\ \mathbf{let}\ F[\,t : \mathit{Seq}(\{s[i] : i \in 1 \,..\, \mathit{Len}(s)\})\,] \ \triangleq \\
\qquad\qquad\qquad\qquad\qquad \mathbf{if}\ t = \langle\,\rangle\ \mathbf{then}\ \langle\,\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{else}\ \ \mathbf{if}\ \mathit{test}(\mathit{Head}(t)) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{then}\ \langle \mathit{Head}(t) \rangle \circ F[\mathit{Tail}(t)] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{else}\ \ F[\mathit{Tail}(t)] \\
\qquad\qquad\qquad\qquad \mathbf{in}\ \ F[s]
\end{array}
$$

$$
\mathit{SubSeq}(s, m, n) \ \triangleq\ [i \in 1 \,..(1 + n - m) \mapsto s[i + m - 1]]
$$

Figure 7: Module *Sequences*

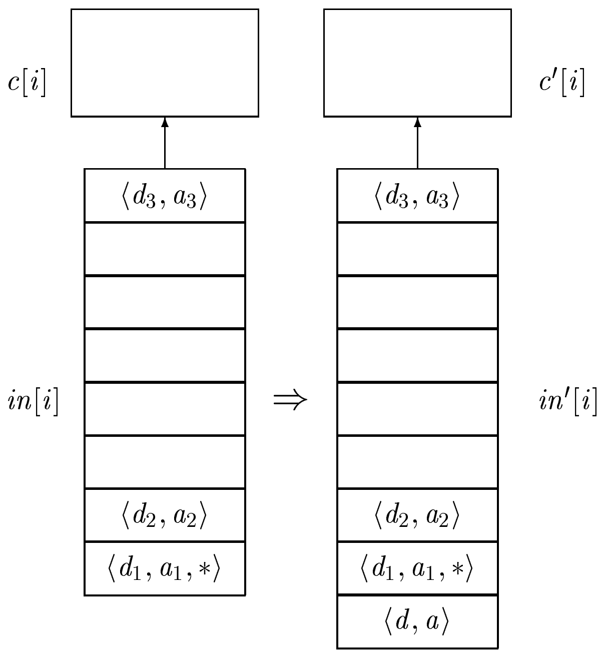

Figure 8: The actions *MemWrite* and *MemRead* both add to $in[i]$

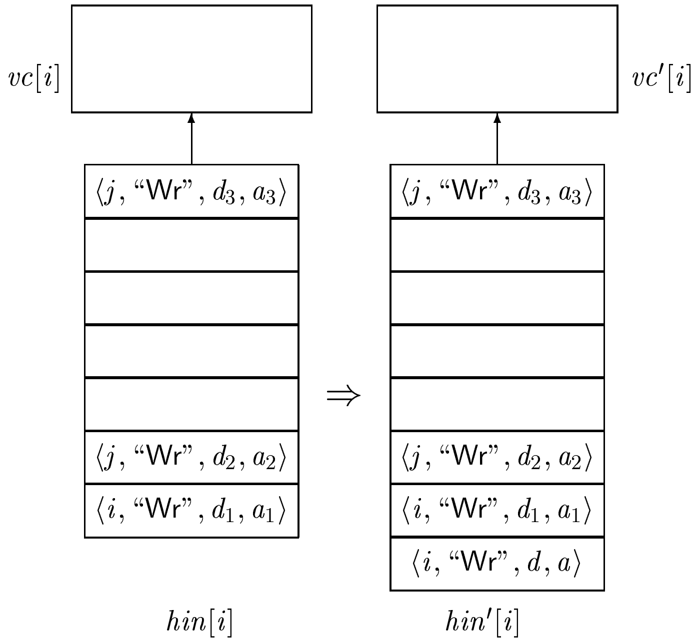

Figure 9: The virtual *MemWrite*

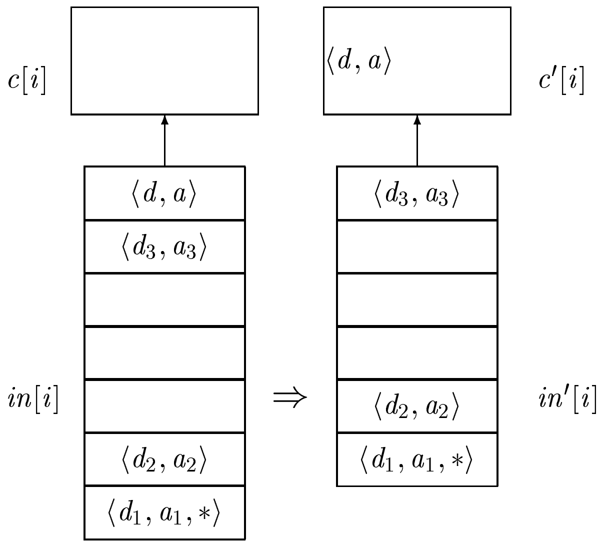

Figure 10: *CacheUpdate*

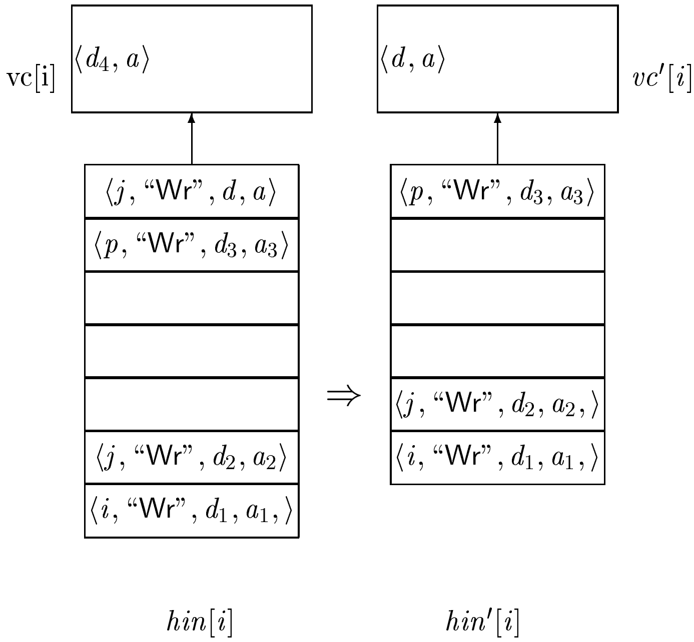

Figure 11: *HCacheUpdate*

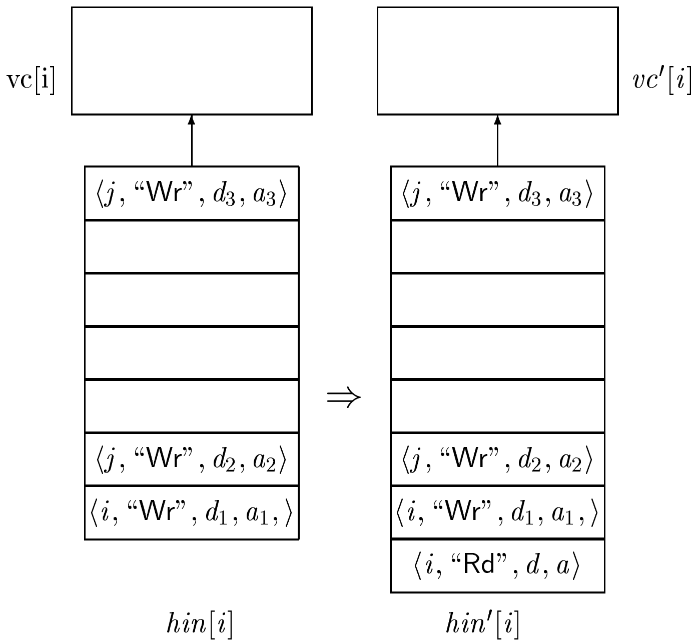

Figure 12: *HMemRead*

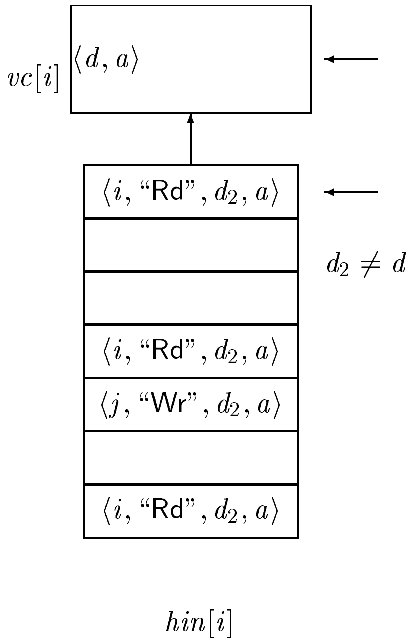

Figure 13: What must **not** happen

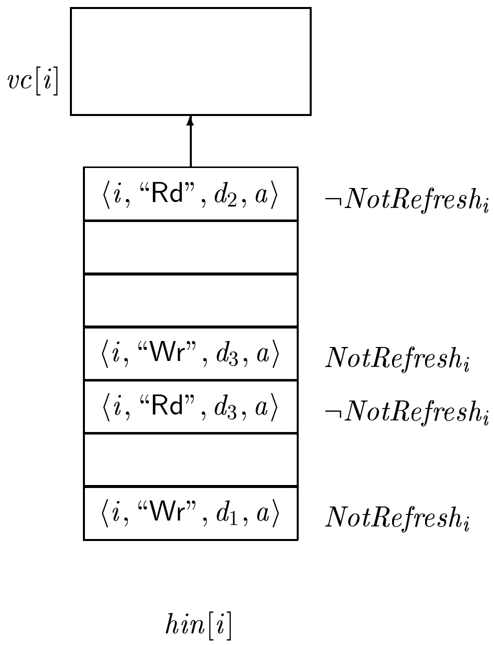

Figure 14: Illustrating $NotRefresh_i$

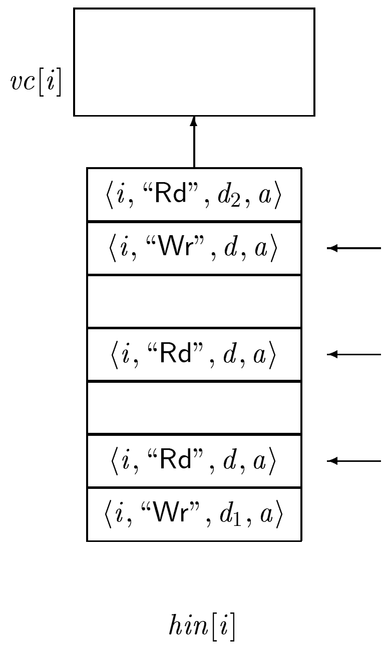

Figure 15: $MemWritesFromTo(2, 7, i)$

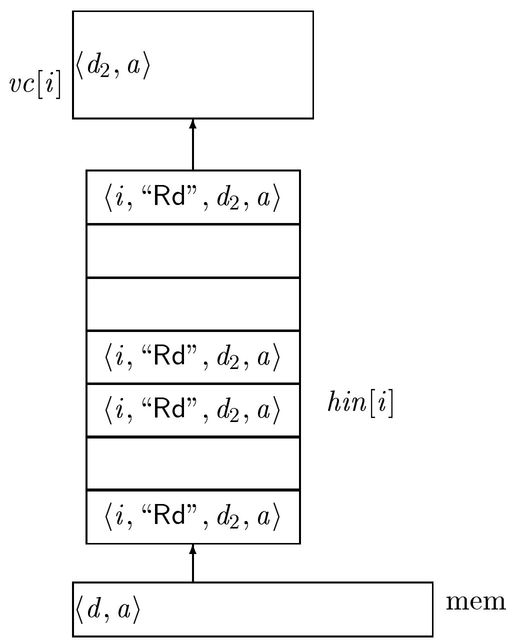

Figure 16: Also to be avoided

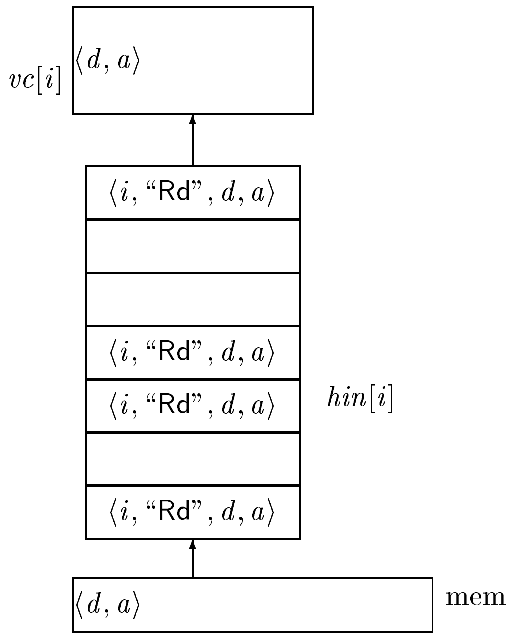

Figure 17: That's better!

situation would also eventually lead to the bad situation excluded by *FirstIsVC*. Finally, if there are only cache refreshes of address a in $in[i]$, and no *NotRefresh*es, then both $vc[i]$ and all the refreshes must contain the same *datum* as *mem* for that address a. This situation is illustrated in Figure 17, and described by the predicate $NoMemWrite(i)$ in the invariant.

Putting this all together, the invariant is that

- Refreshes between adjacent *NotRefresh* writes must be the same as the first such;
- When there are only refreshes in $in[i]$, they must contain the same data value as held for that address in $vc[i]$ and in *mem*;
- Otherwise, refreshes must have the same value as $vc[i]$ until the first "Wr"-write, and as *mem* beyond the last one.

This is expressed by the formula

$$
\begin{aligned}
CCInv.3(p) \;\triangleq\;& \land\ BetweenMemWrites(p) \\
& \land\ \forall\, a \in Addr\ : \\
& \qquad \lor\ NoMemWrite(a, p) \\
& \qquad \lor\ \land\ LastIsMem(a, p) \\
& \qquad\quad\ \land\ FirstIsVC(a, p)
\end{aligned}
$$

As usual, there are a couple of other minor 'type-conditions' that must be stated in the invariant. The entire invariant is described within the proof. The intuition of how the situation in the queue $hin[i]$ evolves is crucial to decomposing the proof of the invariant into proofs of its individual clauses.

4 The Proof

The true taste of success for any invariant comes when we successfully use it to prove the safety property of *CCache*. This is straightforward. The harder part is to prove that the invariant is truly invariant.

The liveness properties of the algorithm are trivial (not our fault–they were imposed by Gerth), and so the liveness part of the proof is routine and mathematically almost empty. One piece, the weak fairness of the cache update operation in *CCache*, is formally similar to the proof of the weak fairness of an abstract buffer *pop* operation in [Lad95]. The abstract operation is enabled before the concrete operation is, and the concrete operation must be performed, invisibly to the specification, on "Rd"-writes numerous times in order to bring the "Wr"-write to the *Head* of $hin[i]$, thus enabling the concrete update of $vc[i]$ which corresponds to the abstract update $CC(cc, cin, cout).CacheUpdate$. This proof thus requires the use of TLA rule WF2, of which the fourth premiss is justified by a temporal logic rule called the Bridge Rule ([Lad95, Page 46]), one of whose hypotheses is the conclusion of the Lattice Rule, of which the premiss is justified in turn by WF1, of which the premises are directly proved. In contrast to the proof in [Lad95], the safety proof of *LCimplCC* is not trivial, although it is straightforward, as we have seen from our justification of the invariant. Now to the proof. Have fun!

$\langle 1 \rangle 1.$ $LC.Spec \Rightarrow (\exists\, cc, cin, cout : CC(cc, cin, cout).Spec)$

PROOF:

LET: We add history variables: *vc* is the virtual cache; *hout* is *out*, with more information carried along as required by $CCache(cc, cin, cout)$; *hin* contains not only the writes from *in* but also whether the writes were the result of a $MemWrite(i)$ for some $i \in Proc$ (the second entry of this element of $hin[p]$ is "Wr"), or the result of a cache refresh, $MemRead(a, p)$ (the second entry of this element of $hin[p]$ is "Rd").

We define new operations which are identical to the LazyCache operations for the LazyCache variables, but which alter the history variables as we wish.

$$
\begin{array}{l}
vc, hin, hout \;:\; \text{VARIABLE} \\
hvars \;\triangleq\; LC.vars \circ \langle vc, hin, hout \rangle \\
new.hin \;\triangleq\; \textbf{case}\ \exists\, i \in Proc, d \in Data, a \in Addr : Read(i, d, a) \rightarrow hin, \\
\qquad\qquad \exists\, i \in Proc, d \in Data, a \in Addr : Write(i, d, a) \rightarrow hin, \\
\qquad\qquad \exists\, i \in Proc : MemWrite(i) \rightarrow \\
\qquad\qquad\qquad [hin \text{ EXCEPT } ![i] = hin[i] \circ \langle \langle i, \text{“Wr”} \rangle \circ Head(out[i]) \rangle], \\
\qquad\qquad \exists\, i \in Proc : CacheUpdate(i) \rightarrow \\
\qquad\qquad\qquad [hin \text{ EXCEPT } ![i] = Tail(hin[i])], \\
\qquad\qquad \exists\, i \in Proc : MemRead(i) \rightarrow \\
\qquad\qquad\qquad [hin \text{ EXCEPT } ![i] = hin[i] \circ \langle \langle i, \text{“Rd”}, \perp, \perp \rangle \rangle], \\
\qquad\qquad \exists\, i \in Proc : CacheInval(i) \rightarrow hin \\
new.vc \;\triangleq\; \textbf{case}\ \exists\, i \in Proc, d \in Data, a \in Addr : Read(i, d, a) \rightarrow vc, \\
\qquad\qquad \exists\, i \in Proc, d \in Data, a \in Addr : Write(i, d, a) \rightarrow vc, \\
\qquad\qquad \exists\, i \in Proc : MemWrite(i) \rightarrow vc, \\
\qquad\qquad \exists\, i \in Proc : CacheUpdate(i) \rightarrow, \\
\qquad\qquad\qquad [vc \text{ EXCEPT } ![i][Head(in[i])[2]] = Head(in[i])[1]], \\
\qquad\qquad \exists\, i \in Proc : MemRead(i) \rightarrow vc, \\
\qquad\qquad \exists\, i \in Proc : CacheInval(i) \rightarrow vc
\end{array}
$$

$$
\begin{array}{lll}
new.hout & \triangleq & \textbf{case}\ \exists\, i \in Proc, d \in Data, a \in Addr\ :\ Read(i,d,a) \rightarrow hout, \\
& & \quad \exists\, i \in Proc, d \in Data, a \in Addr\ :\ Write(i,d,a) \rightarrow \\
& & \qquad\qquad [hout\ \text{EXCEPT}\ ![i] = hout[i] \circ \langle\langle i, \text{"Wr"}, d, a\rangle\rangle], \\
& & \quad \exists\, i \in Proc\ :\ MemWrite(i) \rightarrow [hout\ \text{EXCEPT}\ ![i] = Tail(hout[i])], \\
& & \quad \exists\, i \in Proc\ :\ CacheUpdate(i) \rightarrow hout, \\
& & \quad \exists\, i \in Proc\ :\ MemRead(i) \rightarrow hout, \\
& & \quad \exists\, i \in Proc\ :\ CacheInval(i) \rightarrow hout
\end{array}
$$

$$
\begin{array}{lll}
HInit & \triangleq & \land\ LC.Init \\
& & \land\ vc \in [Proc \rightarrow InitDB] \\
& & \land\ \forall\, i \in Proc\ :\ vc[i] = mem \\
& & \land\ hin = [i \in Proc \mapsto \langle\,\rangle] \\
& & \land\ hout = [i \in Proc \mapsto \langle\,\rangle]
\end{array}
$$

$i, d, a\ :\ \text{CONSTANT}$

$$HRead(i,d,a) \triangleq LC.Read(i,d,a) \land \text{UNCHANGED}\ \langle vc, hin, hout\rangle$$

$$
\begin{array}{lll}
HWrite(i,d,a) & \triangleq & \land\ LC.Write(i,d,a) \\
& & \land\ hout' = [hout\ \text{EXCEPT}\ ![i] = hout[i] \circ \langle\langle i, \text{"Wr"}, d, a\rangle\rangle] \\
& & \land\ \text{UNCHANGED}\ \langle vc, hin\rangle \\
HMemWrite(i) & \triangleq & \land\ LC.MemWrite(i) \\
& & \land\ hin' = [j \in Proc \mapsto hin[j] \circ \langle\langle i, \text{"Wr"}\rangle\rangle] \\
& & \land\ hout' = [hout\ \text{EXCEPT}\ ![i] = Tail(hout[i])] \\
& & \land\ \text{UNCHANGED}\ vc \\
HCacheUpdate(i) & \triangleq & \land\ LC.CacheUpdate(i) \\
& & \land\ hin' = [hin\ \text{EXCEPT}\ ![i] = Tail(hin[i])] \\
& & \land\ vc' = [vc\ \text{EXCEPT}\ ![i][Head(in[i])[2]] = Head(in[i])[1]] \\
& & \land\ \text{UNCHANGED}\ hout \\
HMemRead(i) & \triangleq & \land\ LC.MemRead(i) \\
& & \land\ hin' = [hin\ \text{EXCEPT}\ ![i] = hin[i] \circ \langle\langle i, \text{"Rd"}\rangle\rangle] \\
& & \land\ \text{UNCHANGED}\ vc, hout
\end{array}
$$

$$HCacheInval(i) \triangleq LC.CacheInval(i) \land \text{UNCHANGED}\ \langle vc, hin, hout\rangle$$

$$
\begin{array}{lll}
HOp(i) & \triangleq & \lor\ \exists\, d \in Data, a \in Addr\ :\ HRead(i,d,a) \\
& & \lor\ \exists\, d \in Data, a \in Addr\ :\ HWrite(i,d,a) \\
& & \lor\ HMemWrite(i) \\
& & \lor\ HCacheUpdate(i) \\
& & \lor\ HMemRead(i) \\
& & \lor\ HCacheInval(i) \\
HSpec & \triangleq & \land\ HInit \land \Box[\exists\, i \in Proc\ :\ HOp(i)]_{hvars} \\
& & \land\ \forall\, i\ :\in Proc\ :\ \mathrm{WF}_{hvars}(HCacheUpdate(i)) \\
& & \land\ \forall\, i\ :\in Proc\ :\ \mathrm{WF}_{hvars}(HMemWrite(i))
\end{array}
$$

We show, using *Proposition 1 (History Variable)*, that the specification with history variables is logically equivalent to the LazyCache specification.

$\langle 2\rangle 1.$ $LC.Spec \equiv \boldsymbol{\exists}\, vc, hin, hout\ :\ HSpec$

PROOF:

In order to unpack the `case` statement unambiguously, we show first that all the LazyCache operations are mutually exclusive. The step will then follow directly from the History Variable theorem.

$\langle 3 \rangle 1.$ $LC.Read(i, d, a) \Rightarrow \land \neg LC.Write(i, d, a)$
$\quad \land \neg LC.MemWrite(i)$
$\quad \land \neg LC.CacheUpdate(i)$
$\quad \land \neg LC.MemRead(i)$
$\quad \land \neg LC.CacheInval(i)$

PROOF:

$\langle 4 \rangle 1.$ $LC.Read(i, d, a) \Rightarrow \neg LC.Write(i, d, a)$

$\langle 5 \rangle 1.$ $LC.Read(i, d, a) \Rightarrow ChanOp(ch[i], \langle i, \text{“Rd”}, d, a \rangle)$

PROOF: Immediate from the definition. □

$\langle 5 \rangle 2.$ $ChanOp(ch[i], \langle i, \text{“Rd”}, d, a \rangle) \Rightarrow \neg ChanOp(ch[i], \langle i, \text{“Wr”}, d, a \rangle)$

PROOF: Immediate from the definition in *ChannelInterface*. □

$\langle 5 \rangle 3.$ $\neg ChanOp(ch[i], \langle i, \text{“Wr”}, d, a \rangle) \Rightarrow \neg LC.Write(i, d, a)$

PROOF: Immediate from the definition of $LC.Write$, by contraposition. □

$\langle 5 \rangle 4.$ Q.E.D.

By Boolean logic from $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, and $\langle 5 \rangle 3$. □

$\langle 4 \rangle 2.$ $LC.Read(i, d, a) \Rightarrow \neg LC.MemWrite(i)$

$\langle 5 \rangle 1.$ $LC.Read(i, d, a) \Rightarrow in' = in$

PROOF: Immediate from the definition of $LC.Read(i, d, a)$. □

$\langle 5 \rangle 2.$ $in' = in \Rightarrow \neg(in' = [j \in Proc \mapsto in[j] \circ \textbf{if } j = i \textbf{ then } \langle Head(out[i]) \circ \langle \text{“*”} \rangle \rangle \ ) \quad \textbf{else } \langle Head(out[i]) \rangle \ ]$

PROOF: Immediate from properties of $\circ$. □

$\langle 5 \rangle 3.$ $\neg(in' = [j \in Proc \mapsto in[j] \circ \textbf{if } j = i \textbf{ then } \langle Head(out[i]) \circ \langle \text{“*”} \rangle \rangle \ ) \quad \textbf{else } \langle Head(out[i]) \rangle \ ]$
$\quad \Rightarrow \neg LC.MemWrite(i)$

PROOF: Immediate from the definition of $LC.MemWrite$. □

$\langle 5 \rangle 4.$ Q.E.D.

PROOF: Follows by Boolean logic from $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, and $\langle 5 \rangle 3$. □

$\langle 4 \rangle 3.$ $LC.Read(i, d, a) \Rightarrow \neg LC.CacheUpdate(i)$

$\langle 5 \rangle 1.$ $LC.Read(i, d, a) \Rightarrow in' = in$

PROOF: Immediate from the definition of $LC.Read(i, d, a)$. □

$\langle 5 \rangle 2.$ $in' = in \Rightarrow \neg(in' = [in \text{ EXCEPT } ![i] = Tail(in[i])])$

PROOF: Immediate. □

$\langle 5 \rangle 3.$ $\neg(in' = [in \text{ EXCEPT } ![i] = Tail(in[i])]) \Rightarrow \neg LC.CacheUpdate(i)$

PROOF: Immediate from the definition of $LC.CacheUpdate$. □

$\langle 5 \rangle 4.$ Q.E.D.

PROOF: Follows by Boolean logic from $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, and $\langle 5 \rangle 3$. □

$\langle 4 \rangle 4.$ $LC.Read(i, d, a) \Rightarrow \neg LC.MemRead(i)$

$\langle 5 \rangle 1.$ $LC.Read(i, d, a) \Rightarrow in' = in$

PROOF: Immediate from the definition of $LC.Read(i, d, a)$. □

$\langle 5 \rangle 2.$ $in' = in \Rightarrow \neg(\exists\, a \in Addr : in' = [in \text{ EXCEPT } ![i] = in[i] \circ \langle \langle mem[a], a \rangle \rangle])$

PROOF: Immediate. □

$\langle 5 \rangle 3.$ $\neg(\exists\, a \in Addr : in' = [in \text{ EXCEPT } ![i] = in[i] \circ \langle \langle mem[a], a \rangle \rangle])$
$\quad \Rightarrow \neg LC.MemRead(i)$

PROOF: Immediate from the definition of $LC.MemRead$. □

$\langle 5 \rangle 4.$ Q.E.D.

PROOF: Follows by Boolean logic from $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, and $\langle 5 \rangle 3$. □

$\langle 4 \rangle 5.$ $LC.Read(i, d, a) \Rightarrow \neg LC.CacheInval(i)$

$\langle 5 \rangle 1.$ $LC.Read(i, d, a) \Rightarrow ChanOp(ch[i], \langle i, \text{“Rd”}, d, a \rangle)$

PROOF: Immediate from the definition of $LC.Read(i, d, a)$. □

$\langle 5 \rangle 2.$ $ChanOp(ch[i], \langle i, \text{“Rd”}, d, a \rangle) \Rightarrow \neg \text{UNCHANGED } ch$

PROOF: Immediate from the definition of $ChanOp$. □

$\langle 5 \rangle 3.$ $\neg$UNCHANGED $ch \Rightarrow \neg LC.CacheInval(i)$

PROOF: Immediate from the definition of $LC.CacheInval$. □

$\langle 5 \rangle 4.$ Q.E.D.

PROOF: Follows by Boolean logic from $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, and $\langle 5 \rangle 3$. □

$\langle 4 \rangle 6.$ Q.E.D.

Follows by Boolean Logic from $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$. □

$\langle 3 \rangle 2.$ $LC.Write(i, d, a) \Rightarrow \begin{array}{l} \land \neg LC.Read(i, d, a) \\ \land \neg LC.MemWrite(i) \\ \land \neg LC.CacheUpdate(i) \\ \land \neg LC.MemRead(i) \\ \land \neg LC.CacheInval(i) \end{array}$

PROOF: Similar to that of $\langle 3 \rangle 1$. □

$\langle 3 \rangle 3.$ $LC.MemWrite(i, d, a) \Rightarrow \begin{array}{l} \land \neg LC.Read(i, d, a) \\ \land \neg LC.Write(i, d, a) \\ \land \neg LC.CacheUpdate(i) \\ \land \neg LC.MemRead(i) \\ \land \neg LC.CacheInval(i) \end{array}$

PROOF: Similar to that of $\langle 3 \rangle 1$. □

$\langle 3 \rangle 4.$ $LC.CacheUpdate(i) \Rightarrow \begin{array}{l} \land \neg LC.Read(i, d, a) \\ \land \neg LC.Write(i, d, a) \\ \land \neg LC.MemWrite(i) \\ \land \neg LC.MemRead(i) \\ \land \neg LC.CacheInval(i) \end{array}$

PROOF: Similar to that of $\langle 3 \rangle 1$. □

$\langle 3 \rangle 5.$ $LC.MemRead(i) \Rightarrow \begin{array}{l} \land \neg LC.Read(i, d, a) \\ \land \neg LC.Write(i, d, a) \\ \land \neg LC.MemWrite(i) \\ \land \neg LC.CacheUpdate(i) \\ \land \neg LC.CacheInval(i) \end{array}$

PROOF: Similar to that of $\langle 3 \rangle 1$. □

$\langle 3 \rangle 6.$ $LC.CacheInval(i) \Rightarrow \begin{array}{l} \land \neg LC.Read(i, d, a) \\ \land \neg LC.Write(i, d, a) \\ \land \neg LC.MemWrite(i) \\ \land \neg LC.CacheUpdate(i) \\ \land \neg LC.MemRead(i) \end{array}$

PROOF: Similar to that of $\langle 3 \rangle 1$. □

$\langle 3 \rangle 7.$ Q.E.D.

PROOF: Follows directly from the conclusion of *Proposition 1 (History Variable)* with

$\mathcal{B}_1 \triangleq \mathcal{A}_1 = \exists\, i : LC.MemWrite(i),$

$\mathcal{B}_2 \triangleq \mathcal{A}_2 = \exists\, i : LC.CacheUpdate(i),$

$\mathcal{B}_3 \triangleq \mathcal{A}_3 = \exists\, i : LC.MemRead(i),$

$\mathcal{A}_4 \triangleq \exists\, i, d, a : LC.Read(i, d, a),$

$\mathcal{A}_5 \triangleq \exists\, i, d, a : LC.Write(i, d, a),$

$\mathcal{A}_6 \triangleq \exists\, i : LC.CacheInval(i),$

$h \triangleq \langle vc, hin, hout \rangle,$

$g \triangleq \langle new.vc, new.hin, new.hout \rangle.$

Steps 1 and 2 of the theorem follow immediately from the binding just given, and Step 3 follows with $Inv = $ TRUE from $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 5$, $\langle 3 \rangle 3$, $\langle 3 \rangle 4$, and $\langle 3 \rangle 6$. The theorem follows finally by using the definition of tuples to select the elements of the tuple h where required. □

We show that the LazyCache-with-history specification implies the CCache specification. (The theorem that LazyCache implements CCache then will follow immediately using ⟨2⟩1, the next step and the transitivity of implication.)

⟨2⟩2. $\exists\, c, in, out, mem, ch, vc, hin, hout \;:\; \begin{array}{l} \land\ HInit \\ \land\ \Box[\exists\, i \in Proc \;:\; HOp(i)]_{hvars} \\ \land\ \forall\, i \;:\in Proc \;:\; \mathrm{WF}_{hvars}(HCacheUpdate(i)) \\ \land\ \forall\, i \;:\in Proc \;:\; \mathrm{WF}_{hvars}(HMemWrite(i))) \end{array}$

$\Rightarrow \exists\, cc, cin, cout, ch \;:\; \begin{array}{l} \land\ CC(cc, cin, cout).Init \\ \land\ \Box[\exists\, i \in Proc \;:\; CC(cc, cin, cout).Next(i)]_{\langle cc, cin, cout, ch\rangle} \\ \land\ \forall\, i \;:\in Proc \;:\; \mathrm{WF}_{\langle cc, cin, cout, ch\rangle}(CC(cc, cin, cout).CacheUpdate(i)) \\ \land\ \forall\, i \;:\in Proc \;:\; \mathrm{WF}_{\langle cc, cin, cout, ch\rangle}(CC(cc, cin, cout).MemWrite(i))) \end{array}$

PROOF:

LET: The refinement mapping is

$$\begin{array}{l} \overline{cc} \leftarrow vc \\ \overline{cin} \leftarrow [i \in Proc \mapsto SelectSeq(hin[i], P[2] \neq \text{“}\mathsf{Rd}\text{”})] \\ \overline{cout} \leftarrow hout \\ \overline{ch} \leftarrow ch \end{array}$$

$j, p \;:\;$ CONSTANT

The following predicate is needed to state the invariant $CCInv.3(p)$

$$\forall\, i \in Proc \;:\; NotRefresh_i(j) \;\triangleq\; \begin{array}{l} \land\ j \in 1\,..\,Len(hin[i]) \\ \land\ hin[i][j][2] \neq \text{“}\mathsf{Rd}\text{”} \end{array}$$

The first two conjuncts of the invariant are routine:

$$\begin{array}{lll} CCInv.1(p) & \triangleq & \land\ Len(hin[p]) = Len(in[p]) \\ & & \land\ Len(hout[p]) = Len(out[p]) \\ & & \land\ \forall\, a \in Addr, d \in Data, j \in 1\,..\,Len(in[i]) \;: \\ & & \quad Len(in[i][j]) = 3 \equiv \land\ hin[i][j][1] = i \\ & & \quad \land\ hin[i][j][2] = \text{“}\mathsf{Wr}\text{”} \\ CCInv.2(p) & \triangleq & c[p] \in Restrict(vc[p]) \end{array}$$

The third conjunct requires some auxiliary definitions. First, *NoMemWrite* asserts that there's no element in the $in[p]$ queue that is a write of address a.

$$NoMemWrite(a, p) \;\triangleq\; \begin{array}{l} \land\ p \in Proc \\ \land\ vc[p][a] = mem[a] \\ \land\ \forall\, k \in 1\,..\,Len(in[p]) \;:\; \lor\ in[p][k][2] \neq a \\ \lor\ in[p][k][1] = mem[a] \end{array}$$

But suppose there is an element in the queue $in[p]$ that is a write of address a. The following predicate asserts that the last element in the $in[p]$ queue that is the result of a *MemWrite* of address a contains the same data value as *mem*.

$$
\begin{array}{lll}
\mathit{LastIsMem}(a, p) & \triangleq & \land\ p \in \mathit{Proc} \\
 & & \land\ \exists\, k \in 1 \mathrel{..} \mathit{Len}(\mathit{in}[p]) : \\
 & & \qquad \land\ \mathit{NotRefresh}_p(k) \\
 & & \qquad \land\ \mathit{in}[p][k][1] = \mathit{mem}[a] \\
 & & \qquad \land\ \mathit{in}[p][k][2] = a \\
 & & \qquad \land\ \forall\, j \in k+1 \mathrel{..} \mathit{Len}(\mathit{in}[p]) : \begin{array}[t]{l} \lor\ \mathit{in}[p][j][2] \neq a \\ \lor\ \mathit{in}[p][j][1] = \mathit{mem}[a] \end{array}
\end{array}
$$

The following predicate asserts that all elements in $in[p]$ with address a that occur before the first element that is a $MemWrite$ of address a must have the same data value as lies in the cache $vc[p]$. (Note always $vc[p][a] \in Data$ for all $a \in Addr$.)

$$
\begin{array}{lll}
\mathit{FirstIsVC}(a, p) & \triangleq & \land\ p \in \mathit{Proc} \\
 & & \land\ \exists\, k \in 1 \mathrel{..} \mathit{Len}(\mathit{in}[p]) : \\
 & & \qquad \land\ \mathit{NotRefresh}_p(k) \\
 & & \qquad \land\ \mathit{in}[p][k][2] = a \\
 & & \qquad \land\ \forall\, j \in 1 \mathrel{..} k-1 : \begin{array}[t]{l} \lor\ \mathit{in}[p][j][2] \neq a \\ \lor\ \mathit{in}[p][j][1] = \mathit{vc}[p][a] \end{array}
\end{array}
$$

Between two $MemWrite$s of address a in $in[p]$, say $in[p][m]$ and $in[p][n]$, every element with address a has the same data value as $in[p][m]$.

$m, n\ :\ \textsc{constant}$

$$
\begin{array}{l}
\mathit{MemWritesFromTo}(m, n, p) \ \triangleq \\
\qquad \land\ p \in \mathit{Proc} \\
\qquad \land\ m \in 1 \mathrel{..} \mathit{Len}(\mathit{in}[p]) \\
\qquad \land\ n \in 1 \mathrel{..} \mathit{Len}(\mathit{in}[p]) \\
\qquad \land\ m < n \\
\qquad \land\ \mathit{NotRefresh}_p(m) \\
\qquad \land\ \mathit{NotRefresh}_p(n) \\
\qquad \land\ \forall\, j \in m+1 \mathrel{..} n-1 : \neg \mathit{NotRefresh}_p(j) \\
\qquad \Rightarrow \\
\qquad\qquad \forall\, j \in m+1 \mathrel{..} n-1 : \begin{array}[t]{l} \lor\ \mathit{in}[p][j][2] \neq \mathit{in}[p][m][2] \\ \lor\ \mathit{in}[p][j][1] = \mathit{in}[p][m][1] \end{array}
\end{array}
$$

.... and the invariant will assert this be true for all m and n.

$$
\mathit{BetweenMemWrites}(p) \ \triangleq\ \forall\, m, n \in 1 \mathrel{..} \mathit{Len}(\mathit{in}[p]) : \mathit{MemWritesFromTo}(m, n, p)
$$

Finally, here is the third component of the invariant for process p.

$$
\begin{array}{lll}
\mathit{CCInv}.3(p) & \triangleq & \land\ \mathit{BetweenMemWrites}(p) \\
 & & \land\ \forall\, a \in \mathit{Addr} : \\
 & & \qquad \lor\ \mathit{NoMemWrite}(a, p) \\
 & & \qquad \lor\ \begin{array}[t]{l} \land\ \mathit{LastIsMem}(a, p) \\ \land\ \mathit{FirstIsVC}(a, p) \end{array}
\end{array}
$$

The definition of $CCInv(p)$, the invariant for process p is:

$$CCInv(p) \triangleq CCInv.1(p) \wedge CCInv.2(p) \wedge CCInv.3(p)$$

The definition of the complete invariant is

$$\begin{array}{lcl} CCInv.1 & \triangleq & \forall p \in Proc : CCInv.1(p) \\ CCInv.2 & \triangleq & \forall p \in Proc : CCInv.2(p) \\ CCInv.3 & \triangleq & \forall p \in Proc : CCInv.3(p) \\ CCInv & \triangleq & CCInv.1 \wedge CCInv.2 \wedge CCInv.3 \end{array}$$

First, some straightforward logical observations concerning the invariant.

Everything but an $HMemWrite(i)$ leaves data structures for all other processes but i alone, as also mem.

PROOF:
$\langle 3 \rangle 1.$ $HOp(i) \wedge \neg HMemWrite(i) \Rightarrow$
$\quad \forall p \in Proc \setminus \{i\} : \text{UNCHANGED } \langle c[p], in[p], out[p], mem, ch[p], vc[p], hin[p], hout[p] \rangle$
PROOF: Immediate from the definitions of *HRead*, *HWrite*, *HMemWrite*, *HCacheUpdate*, *HMemRead*, and *HCacheInval*. □

A $HMemWrite(i)$ leaves $c[p], out[p], ch[p], vc[p]$ and $hout[p]$ alone, for all $p \in Proc$ such that $p \neq i$.

$\langle 3 \rangle 2.$ $HMemWrite(i) \Rightarrow \forall p \in Proc \setminus \{i\} : \text{UNCHANGED } \langle c[p], out[p], ch[p], vc[p], hout[p] \rangle$
PROOF: Immediate from the definitions of *HRead*, *HWrite*, *HMemWrite*, *HCacheUpdate*, *HMemRead*, and *HCacheInval*. □

Thus everything except an $HMemWrite(i)$ leaves the variables in $CCInv(p)$ alone, hence $CCInv(p)'$ is equivalent to $CCInv(p)$.

$\langle 3 \rangle 3.$ $HOp(i) \wedge \neg HMemWrite(i) \Rightarrow$
$\quad \forall p \in Proc \setminus \{i\} : CCInv(p) \equiv CCInv'(p)$
PROOF: Immediate from $\langle 3 \rangle 1$ and the definition of $CCInv(p)$. □

The initial state of LazyCache implies the initial state of CCache.

$\langle 3\rangle 4.\ \ HInit \Rightarrow \overline{CC(cc, cin, cout).Init}$

PROOF:

$\langle 4\rangle 1.\ \ \land\ c \in [Proc \rightarrow InitDB]$
$\qquad \land\ mem \in InitDB$
$\qquad \land\ \forall\, i \in Proc\ :\ \land\ c[i] \in Restrict(mem)$
$\qquad\qquad\qquad\qquad \land\ in[i] = \langle\,\rangle$
$\qquad\qquad\qquad\qquad \land\ out[i] = \langle\,\rangle$
$\qquad \land\ vc \in [Proc \rightarrow InitDB]$
$\qquad \land\ \forall\, i \in Proc\ :\ vc[i] = mem$
$\qquad \land\ hin = [i \in Proc \mapsto \langle\,\rangle]$
$\qquad \land\ hout = [i \in Proc \mapsto \langle\,\rangle]$
$\qquad\quad \Rightarrow \land\ vc \in [Proc \rightarrow InitDB]$
$\qquad\qquad \land\ \forall\, i, j \in Proc\ :\ vc[i] = vc[j]$
$\qquad\qquad \land\ \overline{cin} = [i \in Proc \mapsto \langle\,\rangle]$
$\qquad\qquad \land\ \overline{cout} = [i \in Proc \mapsto \langle\,\rangle]$

PROOF:

$\langle 5\rangle 1.\ \ \forall\, i \in Proc\ :\ (vc[i] = mem)$
$\qquad \Rightarrow \forall\, i, j \in Proc\ :\ vc[i] = vc[j]$

PROOF: Immediate from the logic of equality. □

$\langle 5\rangle 2.\ \ (\forall\, i \in Proc\ :\ hin = [i \in Proc \mapsto \langle\,\rangle]) \Rightarrow (\forall\, i \in Proc\ :\ cin = [i \in Proc \mapsto \langle\,\rangle])$

PROOF: Follows immediately from the definition of $cin[i]$, and that $SelectSeq(\langle\,\rangle, P[2] \neq \text{“Rd”}) = \langle\,\rangle$. □

$\langle 5\rangle 3.\ \ (\forall\, i \in Proc\ :\ out[i] = \langle\,\rangle) \Rightarrow \overline{cout} = [i \in Proc \mapsto \langle\,\rangle]$

PROOF: Immediate from the definition of $hout[i]$. □

$\langle 5\rangle 4.$ Q.E.D.

PROOF: Immediate by propositional logic from $\langle 5\rangle 1$, $\langle 5\rangle 2$, and $\langle 5\rangle 3$. □

$\langle 4\rangle 2.$ Q.E.D.

PROOF: Immediate from $\langle 4\rangle 1$ and the definitions of $HInit$, $\overline{CC(cc, cin, cout).Init}$. □

The invariant follows from the initial conditions and the safety property.

$\langle 3\rangle 5.$ $HInit \land \Box[\exists\, i \in Proc : HOp(i)]_{hvars} \Rightarrow \Box CCInv$

The invariant follows from the initial conditions.

$\langle 4\rangle 1.$ $HInit \Rightarrow CCInv$

PROOF:

$\langle 5\rangle 1.$ $HInit \Rightarrow \land\ \forall\, i \in Proc : Len(hin[i]) = 0$
$\qquad\qquad\qquad \land\ \forall\, i \in Proc : Len(in[i]) = 0$
$\qquad\qquad\qquad \land\ \forall\, i \in Proc : Len(hout[i]) = 0$
$\qquad\qquad\qquad \land\ \forall\, i \in Proc : Len(out[i]) = 0$

PROOF: Immediate by definition of *HInit*. □

$\langle 5\rangle 2.$ $HInit \Rightarrow CCInv.1$

PROOF: Immediate from $\langle 5\rangle 1$. □

$\langle 5\rangle 3.$ $HInit \Rightarrow \forall\, i \in Proc : (vc[i] = mem) \land (c[i] = Restrict(mem))$

PROOF: The consequent is part of the definition of *HInit*. □

$\langle 5\rangle 4.$ $HInit \Rightarrow CCInv.2$

PROOF: Immediate from $\langle 5\rangle 3$. □

$\langle 5\rangle 5.$ $HInit \Rightarrow CCInv.3$

PROOF:

$\langle 6\rangle 1.$ $\land\ HInit$
$\qquad \land\ p \in Proc$
$\qquad\quad \Rightarrow \land\ BetweenMemWrites(p)$
$\qquad\qquad\ \ \land\ \forall\, a \in Addr : NoMemWrite(a, p)$

PROOF:

$\langle 7\rangle 1.$ $\land\ HInit$
$\qquad \land\ p \in Proc$
$\qquad\quad \Rightarrow 1 \,..\, Len(in[p]) = \emptyset$

PROOF: Immediate from $in[p] = \langle\, \rangle$.

$\langle 7\rangle 2.$ Q.E.D.

PROOF: $BetweenMemWrites(p)$ follows trivially from $\langle 7\rangle 1$ since both universal quantifiers in the prefix range over $1 \,..\, Len(in[p])$, as does the second conjunct of $NoMemWrite(a, p)$. The first conjunct of $NoMemWrite(a, p)$ follows immediately from $\langle 5\rangle 3$. □

$\langle 6\rangle 2.$ Q.E.D.

PROOF: Immediate from $\langle 6\rangle 1$ and $\langle 3\rangle 3$. □

$\langle 5\rangle 6.$ Q.E.D.

PROOF: Follows by propositional logic from $\langle 5\rangle 2$, $\langle 5\rangle 4$, $\langle 5\rangle 5$ and definition of *CCInv*. □

The invariant is preserved by the steps of LazyCache.

$\langle 4\rangle 2.$ $CCInv \wedge \Box[\exists\, i \in Proc : HOp(i))]_{hvars} \Rightarrow \Box CCInv'$

PROOF:

$\langle 5\rangle 1.$ $CCInv \wedge [\exists\, i \in Proc : HOp(i))]_{hvars} \Rightarrow CCInv'$

PROOF:

$\langle 6\rangle 1.$ $CCInv \wedge hvars' = hvars \Rightarrow CCInv'$

PROOF: Immediate since all the variables of $CCInv$ are in $hvars$. □

First, a general fact from logic and set theory that we shall use without further proof. Let ψ be a TLA formula:

LET: x, y : CONSTANT

$\langle 6\rangle 2.$ $\wedge\ (x \in S \Rightarrow \psi) \equiv (x \in S \setminus \{y\} \Rightarrow \psi) \wedge \psi(y/x)$
$\wedge\ (x \in S \wedge \psi) \equiv (x \in S \setminus \{y\} \wedge \psi) \vee \psi(y/x)$

PROOF: Follows from logic and elementary set theory. □

We split the proof of the invariant into separate steps for each disjunct of HOp.

The invariant is preserved by $HRead$.

$\langle 6\rangle 3.$ $\wedge\ i \in Proc$
$\wedge\ CCInv$
$\wedge\ (\exists\, d \in Data, a \in Addr : HRead(i, d, a))$
$\Rightarrow CCInv'$

PROOF:

$\langle 7\rangle 1.$ $\wedge\ i \in Proc$
$\wedge\ CCInv$
$\wedge\ HRead(i, d, a)$
$\Rightarrow CCInv'$

PROOF: The free variables in, vc, hin, mem, c of invariant $CCInv$ are $HRead$-UNCHANGED variables by the definition of $HRead(i)$. $CCInv'$ follows by the logical laws of equality. □

$\langle 7\rangle 2.$ Q.E.D.

PROOF: $\langle 6\rangle 3$ follows by existential quantification over a and d. □

The invariant is preserved by a $HWrite$

$\langle 6\rangle 4.$ $\wedge\ i \in Proc$
$\wedge\ CCInv$
$\wedge\ (\exists\, d \in Data, a \in Addr : HWrite(i, d, a))$
$\Rightarrow CCInv'$

PROOF:

$\langle 7\rangle 1.$ $\wedge\ i \in Proc$
$\wedge\ CCInv$
$\wedge\ HWrite(i, d, a)$
$\Rightarrow CCInv'$

PROOF:

$\langle 8\rangle 1.$ $i \in Proc \wedge HWrite(i, d, a) \Rightarrow \wedge\ Len(out'[i]) = Len(out[i]) + 1$
$\wedge\ Len(hout'[i]) = Len(hout[i]) + 1$

PROOF: Immediate from the definition of $HWrite$. □

$\langle 8\rangle 2.$ Q.E.D.

PROOF: The conjuncts for out and $hout$ in $CCInv.1$ are preserved by $\langle 8\rangle 1$. The free variables in, vc, hin, mem, c of invariant $CCInv$ are a subset of the $HWrite$-UNCHANGED variables by the definition of $HWrite$. $CCInv'$ follows by the logical laws of equality. □

$\langle 7\rangle 2.$ Q.E.D.

PROOF: $\langle 6\rangle 4$ follows by existential quantification over a and d. □

The invariant is preserved by a $HMemWrite$

$\langle 6\rangle 5.$ $\wedge\ i \in Proc$
$\wedge\ CCInv$
$\wedge\ HMemWrite(i)$
$\Rightarrow CCInv'$

PROOF:

First, some steps that we'll make use of in various places in this proof.

$\langle 7\rangle 1.$ $\wedge\ i \in Proc$
$\wedge\ HMemWrite(i)$
$\wedge\ p \in Proc$
$\Rightarrow \wedge\ Len(hin'[p]) = Len(hin[p]) + 1$
$\wedge\ Len(in'[p]) = Len(in[p]) + 1$
$\wedge\ Len(hout'[p]) = Len(hout[p]) - 1$
$\wedge\ Len(out'[p]) = Len(out[p]) - 1$

PROOF: Immediate from the definition of $HMemWrite$ and the definition of $Tail$. □

$\langle 7\rangle 2.$ $\wedge\ i \in Proc$
$\wedge\ HMemWrite(i)$
$\wedge\ p \in Proc$
$\wedge\ k \in 1 \mathrel{..} Len(in'[p])$
$\wedge\ j \in 1 \mathrel{..} k - 1$
$\Rightarrow j \in 1 \mathrel{..} Len(in[p])$

PROOF: Immediate from $\langle 7\rangle 1$. □

$\langle 7\rangle 3.$ $i \in Proc \wedge HMemWrite(i) \wedge p \in Proc$
$\Rightarrow 1 \mathrel{..} Len(in'[p]) \setminus \{Len(in'[p])\} = 1 \mathrel{..} Len(in[p])$

PROOF: Immediate from $\langle 7\rangle 1$ and definition of $1 \mathrel{..} t$. □

$\langle 7\rangle 4.$ $\wedge\ i \in Proc$
$\wedge\ HMemWrite(i)$
$\wedge\ p \in Proc$
$\wedge\ CCInv.1(p)$
$\wedge\ a \in Addr$
$\wedge\ k \in 1 \mathrel{..} Len(in[p])$
$\Rightarrow \wedge\ in'[p][k] = in[p][k]$
$\wedge\ hin'[p][k] = hin[p][k]$
$\wedge\ vc'[p][a] = vc[p][a]$

PROOF: Immediate from the definition of $HMemWrite$. □

$\langle 7\rangle 5.$ $\wedge\ i \in Proc$
$\wedge\ HMemWrite(i)$
$\wedge\ a \in Addr \setminus \{Head(out[i])[2]\}$
$\Rightarrow \quad mem'[a] = mem[a]$

PROOF: Immediate from the definition of $HMemWrite$. □

$\langle 7\rangle 6.$ $\wedge\ i \in Proc$
$\wedge\ HMemWrite(i)$
$\wedge\ p \in Proc$
$\wedge\ k = Len(in'[p])$
$\Rightarrow NotRefresh'_p(k)$

PROOF: Immediate from the definition of $HMemWrite$. □

The first two conjuncts of the invariant, $CCInv.1$ and $CCInv.2$, are easily seen to be preserved.

$\langle 7 \rangle 7.$ $\land\ i \in Proc$
$\land\ HMemWrite(i)$
$\land\ p \in Proc$
$\land\ CCInv.1(p)$
$\Rightarrow CCInv.1'(p)$

PROOF: The first two conjuncts of $CCInv.1(p)$ follow by equality manipulation from $\langle 7 \rangle 1$, and the third conjunct by $\langle 6 \rangle 2$ and the definition of $HMemWrite$. $\square$

$\langle 7 \rangle 8.$ $\land\ i \in Proc$
$\land\ HMemWrite(i)$
$\land\ p \in Proc$
$\Rightarrow CCInv.2(p) \equiv CCInv.2'(p)$

PROOF: Follows immediately by the definition of $HMemWrite$ since c and vc are UNCHANGED . $\square$

The proof of the third invariant from $HMemWrite$.

$\langle 7 \rangle 9.$ $\land\ i \in Proc$
$\land\ HMemWrite(i)$
$\land\ p \in Proc$
$\land\ CCInv(p)$
$\Rightarrow CCInv.3'(p)$

PROOF:

$BetweenMemWrites$ is preserved. A weakened form of $CCInv.3(p)$ is used as a hypothesis in the proof.

$\langle 8 \rangle 1.$ $\land\ i \in Proc \land HMemWrite(i)$
$\land\ p \in Proc$
$\land\ CCInv.1(p)$
$\land\ BetweenMemWrites(p)$
$\land\ \forall\, a \in Addr\ :\ \lor\ LastIsMem(a, p)$
$\lor\ NoMemWrite(a, p)$
$\Rightarrow BetweenMemWrites'(p)$

PROOF:

The proof is split into two: One proof for $m < n \leq Len(in[p])$ and one for $m \leq Len(in[p])$ and $n = Len(in'[p])$.
First, the split.

$\langle 9 \rangle 1.$ $\land\ i \in Proc \land HMemWrite(i)$
$\land\ p \in Proc$
$\Rightarrow$

$$\begin{aligned} & BetweenMemWrites'(p) \\ \equiv\quad & \land\ \forall\, m, n \in 1\, ..\, Len(in[p])\ :\ MemWritesFromTo'(m, n, p) \\ & \land\ \forall\, m \in 1\, ..\, Len(in[p])\ :\ MemWritesFromTo'(m, Len(in'[p]), p) \end{aligned}$$

PROOF: Follows immediately by $\langle 7 \rangle 1$ and $\langle 6 \rangle 2$ from definition of $1\, ..\, Len(in[p])$ and properties of $<$. $\square$

The step for $m < n \leq Len(in[p])$.

$\langle 9\rangle 2.$ $\land\ i \in \mathit{Proc} \land \mathit{HMemWrite}(i)$
$\land\ p \in \mathit{Proc}$
$\land\ \mathit{CCInv}.1(p)$
$\land\ m \in 1 \mathrel{..} \mathit{Len}(\mathit{in}[p])$
$\land\ n \in 1 \mathrel{..} \mathit{Len}(\mathit{in}[p])$
$\land\ \mathit{MemWritesFromTo}(m, n, p)$
$\quad \Rightarrow \quad \mathit{MemWritesFromTo}'(m, n, p)$

PROOF:

$\langle 10\rangle 1.$ $\land\ i \in \mathit{Proc} \land \mathit{HMemWrite}(i)$
$\land\ \mathit{CCInv}.1(p)$
$\land\ p \in \mathit{Proc}$
$\land\ m \in 1 \mathrel{..} \mathit{Len}(\mathit{in}[p])$
$\land\ n \in 1 \mathrel{..} \mathit{Len}(\mathit{in}[p])$
$\land\ m < n$
$\land\ \mathit{NotRefresh}_p(m)$
$\land\ \mathit{NotRefresh}_p(n)$
$\land\ \forall j \in m+1 \mathrel{..} n-1 : \neg \mathit{NotRefresh}_p(j)$
$\quad \Rightarrow \forall j \in m+1 \mathrel{..} n-1 : \lor\ \mathit{in}[p][j][2] \neq \mathit{in}[p][m][2]$
$\qquad\qquad\qquad\qquad\qquad\quad \lor\ \mathit{in}[p][j][1] = \mathit{in}[p][m][1]$
$\Rightarrow$
$\forall j \in m+1 \mathrel{..} n-1 : \neg \mathit{NotRefresh}'_p(j)$
$\quad \Rightarrow \forall j \in m+1 \mathrel{..} n-1 : \lor\ \mathit{in}'[p][j][2] \neq \mathit{in}'[p][m][2]$
$\qquad\qquad\qquad\qquad\qquad\quad \lor\ \mathit{in}'[p][j][1] = \mathit{in}'[p][m][1]$

PROOF:

The proof is split into two parts which are propositionally combined to yield the proof.

$\langle 11\rangle 1.$ $\land\ i \in \mathit{Proc} \land \mathit{HMemWrite}(i)$
$\land\ \mathit{CCInv}.1(p)$
$\land\ p \in \mathit{Proc}$
$\land\ m \in 1 \mathrel{..} \mathit{Len}(\mathit{in}[p])$
$\land\ n \in 1 \mathrel{..} \mathit{Len}(\mathit{in}[p])$
$\land\ m < n$
$\land\ \mathit{NotRefresh}_p(m)$
$\land\ \mathit{NotRefresh}_p(n)$
$\land\ \forall j \in m+1 \mathrel{..} n-1 : \lor\ \mathit{in}[p][j][2] \neq \mathit{in}[p][m][2]$
$\qquad\qquad\qquad\qquad\quad \lor\ \mathit{in}[p][j][1] = \mathit{in}[p][m][1]$
$\Rightarrow$
$\quad \forall j \in m+1 \mathrel{..} n-1 : \lor\ \mathit{in}'[p][j][2] \neq \mathit{in}'[p][m][2]$
$\qquad\qquad\qquad\qquad\quad \lor\ \mathit{in}'[p][j][1] = \mathit{in}'[p][m][1]$

PROOF:

$\langle 12\rangle 1.$ $\land\ i \in \mathit{Proc} \land \mathit{HMemWrite}(i)$
$\land\ \mathit{CCInv}.1(p)$
$\land\ p \in \mathit{Proc}$
$\land\ m \in 1 \mathrel{..} \mathit{Len}(\mathit{in}[p])$
$\land\ n \in 1 \mathrel{..} \mathit{Len}(\mathit{in}[p])$
$\land\ m < n$
$\land\ \mathit{NotRefresh}_p(m)$
$\land\ \mathit{NotRefresh}_p(n)$
$\land\ j \in m+1 \mathrel{..} n-1$
$\land\ \lor\ \mathit{in}[p][j][2] \neq \mathit{in}[p][m][2]$
$\quad\ \lor\ \mathit{in}[p][j][1] = \mathit{in}[p][m][1]$
$\Rightarrow$

$\land\ j \in m + 1 \,..\, n - 1$
$\land\ \lor\ in'[p][j][2] \neq in'[p][m][2]$
$\quad\ \lor\ in'[p][j][1] = in'[p][m][1]$

PROOF: Follows directly from ⟨7⟩4, using propositional logic. □

⟨12⟩2. Q.E.D.

PROOF: Immediate by universal generalisation over j and quantifier rules. □

⟨11⟩2. $\land\ i \in \mathit{Proc} \land \mathit{HMemWrite}(i)$
$\land\ \mathit{CCInv}.1(p)$
$\land\ p \in \mathit{Proc}$
$\land\ m \in 1 \,..\, \mathit{Len}(in[p])$
$\land\ n \in 1 \,..\, \mathit{Len}(in[p])$
$\land\ m < n$
$\land\ \mathit{NotRefresh}_p(m)$
$\land\ \mathit{NotRefresh}_p(n)$
$\land\ \exists j \in m + 1 \,..\, n - 1 : \mathit{NotRefresh}_p(j)$
$\Rightarrow$
$\quad \exists j \in m + 1 \,..\, n - 1 : \mathit{NotRefresh}'_p(j)$

PROOF:

⟨12⟩1. $\land\ i \in \mathit{Proc} \land \mathit{HMemWrite}(i)$
$\land\ \mathit{CCInv}.1(p)$
$\land\ p \in \mathit{Proc}$
$\land\ m \in 1 \,..\, \mathit{Len}(in[p])$
$\land\ n \in 1 \,..\, \mathit{Len}(in[p])$
$\land\ m < n$
$\land\ \mathit{NotRefresh}_p(m)$
$\land\ \mathit{NotRefresh}_p(n)$
$\land\ j \in m + 1 \,..\, n - 1$
$\land\ \mathit{NotRefresh}_p(j)$
$\Rightarrow$
$\quad \land\ j \in m + 1 \,..\, n - 1$
$\quad \land\ \mathit{NotRefresh}'_p(j)$

PROOF:

⟨13⟩1. $\land\ i \in \mathit{Proc} \land \mathit{HMemWrite}(i)$
$\land\ \mathit{CCInv}.1(p)$
$\land\ p \in \mathit{Proc}$
$\land\ m \in 1 \,..\, \mathit{Len}(in[p])$
$\land\ n \in 1 \,..\, \mathit{Len}(in[p])$
$\land\ m < n$
$\land\ \mathit{NotRefresh}_p(m)$
$\land\ \mathit{NotRefresh}_p(n)$
$\land\ j \in m + 1 \,..\, n - 1$
$\land\ j \in 1 \,..\, \mathit{Len}(hin[p])$
$\land\ hin[p][j][2] \neq$ “Rd”
$\Rightarrow$
$\quad \land\ j \in 1 \,..\, \mathit{Len}(hin'[p])$
$\quad \land\ hin'[p][j][2] \neq$ “Rd”

PROOF: The first conjunct of the conclusion follows by set theory and ⟨7⟩1 and the second by ⟨7⟩4. □

⟨13⟩2. Q.E.D.

PROOF: Immediate by propositional logic and definition of *NotRefresh*. □

⟨12⟩2. Q.E.D.

PROOF: Immediate by existential generalisation over j. □

⟨11⟩3. Q.E.D.

PROOF: Follows from ⟨11⟩1 and ⟨11⟩2 by propositional logic: $((A \land (D \Rightarrow E)) \Rightarrow$

$(F \Rightarrow G))$ follows from $(A \land E \Rightarrow G) \land (A \land \neg D \Rightarrow \neg F)$. □

$\langle 10 \rangle 2$. Q.E.D.

PROOF: Immediate from $\langle 10 \rangle 1$ using definition of $MemWritesFromTo$ and propositional logic: the consequent of $\langle 10 \rangle 1$ is the consequent of $MemWritesFromTo'(m, n, p)$. □

The second part of the proof for $m < n = Len(in'[i])$ uses other clauses of $CCInv.3(p)$.

$$\begin{array}{l}
\langle 9 \rangle 3.\ \land\ i \in Proc \land HMemWrite(i) \\
\quad \land\ p \in Proc \\
\quad \land\ m \in 1 \,..\, Len(in[p]) \\
\quad \land\ \forall\, a \in Addr\ : \\
\qquad \lor\ LastIsMem(a, p) \\
\qquad \lor\ NoMemWrite(a, p) \\
\quad \Rightarrow \land\ p \in Proc \\
\qquad \land\ m \in 1 \,..\, Len(in'[p]) \\
\qquad \land\ Len(in'[p]) \in 1 \,..\, Len(in'[p]) \\
\qquad \land\ m < Len(in'[p]) \\
\qquad \land\ NotRefresh_p(m) \\
\qquad \land\ NotRefresh_p(Len(in'[p])) \\
\qquad \land\ \forall\, j \in m+1 \,..\, Len(in'[p]) - 1\ :\ \neg NotRefresh_p(j) \\
\qquad \Rightarrow \\
\qquad\quad \forall\, j \in m+1 \,..\, Len(in'[p]) - 1\ :\ \lor\ in[p][j][2] \neq in[p][m][2] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ \lor\ in[p][j][1] = in[p][m][1]
\end{array}$$

PROOF:

$$\begin{array}{l}
\langle 10 \rangle 1.\ \land\ i \in Proc \land HMemWrite(i) \\
\quad \land\ p \in Proc \\
\quad \land\ m \in 1 \,..\, Len(in[p]) \\
\quad \land\ NotRefresh_p(m) \\
\quad \land\ \forall\, j \in m+1 \,..\, Len(in'[p]) - 1\ :\ \neg NotRefresh_p(j) \\
\quad \land\ \forall\, a \in Addr\ : \\
\qquad \lor\ LastIsMem(a, p) \\
\qquad \lor\ NoMemWrite(a, p) \\
\quad \Rightarrow \forall\, j \in m+1 \,..\, Len(in'[p]) - 1\ :\ \lor\ in[p][j][2] \neq in[p][m][2] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ \lor\ in[p][j][1] = in[p][m][1]
\end{array}$$

PROOF:

$$\begin{array}{l}
\langle 11 \rangle 1.\ \land\ i \in Proc \land HMemWrite(i) \\
\quad \land\ p \in Proc \\
\quad \land\ m \in 1 \,..\, Len(in[p]) \\
\quad \land\ NotRefresh_p(m) \\
\quad \land\ \forall\, j \in m+1 \,..\, Len(in'[p]) - 1\ :\ \neg NotRefresh_p(j) \\
\quad \land\ \lor\ LastIsMem(in[p][m][2], p) \\
\qquad \lor\ NoMemWrite(in[p][m][2], p) \\
\quad \Rightarrow \forall\, j \in m+1 \,..\, Len(in'[p]) - 1\ :\ \lor\ in[p][j][2] \neq in[p][m][2] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ \lor\ in[p][j][1] = in[p][m][1]
\end{array}$$

PROOF:

$$\begin{array}{l}
\langle 12 \rangle 1.\ \land\ i \in Proc \land HMemWrite(i) \\
\quad \land\ p \in Proc \\
\quad \land\ m \in 1 \,..\, Len(in[p]) \\
\quad \land\ NoMemWrite(in[p][m][2], p) \\
\quad \Rightarrow \\
\qquad \forall\, j \in m+1 \,..\, Len(in[p])\ :\ \lor\ in[p][j][2] \neq in[p][m][2] \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \lor\ in[p][j][1] = in[p][m][1]
\end{array}$$

PROOF: The consequent follows by restricting the quantifier in the third conjunct of $NoMemWrite(in[p][m][2], p)$ to $m{+}1\,..\,Len(in[p])$ instead of $1\,..\,Len(in[p])$ □

$\langle 12\rangle 2.$ $\land\ i \in Proc \land HMemWrite(i)$
$\land\ p \in Proc$
$\land\ m \in 1\,..\,Len(in[p])$
$\land\ NotRefresh_p(m)$
$\land\ LastIsMem(in[p][m][2], p)$
$\land\ \forall j \in m+1\,..\,Len(in[p]) : \neg NotRefresh_p(j)$
$\Rightarrow$

$$\forall j \in m+1\,..\,Len(in[p]) : \begin{array}{l} \lor\ in[p][j][2] \neq in[p][m][2] \\ \lor\ in[p][j][1] = in[p][m][1] \end{array}$$

PROOF:

LET: k : CONSTANT

$\langle 13\rangle 1.$ $\land\ i \in Proc \land HMemWrite(i)$
$\land\ p \in Proc$
$\land\ m \in 1\,..\,Len(in[p])$
$\land\ NotRefresh_p(m)$
$\land\ k \in 1\,..\,Len(in[p])$
$\land\ NotRefresh_p(k)$
$\land\ in[p][k][1] = mem[in[p][m][2]]$
$\land\ in[p][k][2] = in[p][m][2]$
$\land\ \forall j \in k+1\,..\,Len(in[p])$:
$\lor\ in[p][j][2] \neq in[p][m][2]$
$\lor\ in[p][j][1] = mem[in[p][m][2]]$
$\land\ \forall j \in m+1\,..\,Len(in[p]) : \neg NotRefresh_p(j)$
$\Rightarrow$
$\forall j \in m+1\,..\,Len(in[p])$:
$\lor\ in[p][j][2] \neq in[p][m][2]$
$\lor\ in[p][j][1] = in[p][m][1]$

PROOF:

We split into three cases: $k < m, k = m, k > m$. The latter can't hold, which we show first.

$\langle 14\rangle 1.$ $\land\ i \in Proc \land HMemWrite(i)$
$\land\ p \in Proc$
$\land\ m \in 1\,..\,Len(in[p])$
$\land\ NotRefresh_p(m)$
$\land\ k \in 1\,..\,Len(in[p])$
$\land\ NotRefresh_p(k)$
$\land\ \forall j \in m+1\,..\,Len(in[p]) : \neg NotRefresh_p(j)$
$\Rightarrow k \leq m$

PROOF:

$\langle 15\rangle 1.$ $\land\ i \in Proc \land HMemWrite(i)$
$\land\ p \in Proc$
$\land\ m \in 1\,..\,Len(in[p])$
$\land\ NotRefresh_p(m)$
$\land\ k \in 1\,..\,Len(in[p])$
$\land\ NotRefresh_p(k)$
$\land\ k > m$
$\Rightarrow \exists j \in m+1\,..\,Len(in[p]) : NotRefresh_p(j)$

PROOF:

$\langle 16\rangle 1.$ $\wedge\ i \in Proc \wedge HMemWrite(i)$
$\wedge\ p \in Proc$
$\wedge\ m \in 1\,..\,Len(in[p])$
$\wedge\ NotRefresh_p(m)$
$\wedge\ k \in 1\,..\,Len(in[p])$
$\wedge\ NotRefresh_p(k)$
$\wedge\ k > m$
$\Rightarrow \wedge\ k \in m+1\,..\,Len(in[p])$
$\quad \wedge\ NotRefresh_p(k)$

PROOF: Immediate from properties of $>$. □

$\langle 16\rangle 2.$ Q.E.D.

PROOF: Immediate by existential generalisation over k. □

$\langle 15\rangle 2.$ Q.E.D.

PROOF: Immediate from $\langle 15\rangle 1$ by contraposition of $k > m$ with the consequent, and properties of $<$. □

Now the $k = m$ case.

$\langle 14\rangle 2.$ $\wedge\ i \in Proc \wedge HMemWrite(i)$
$\wedge\ p \in Proc$
$\wedge\ m \in 1\,..\,Len(in[p])$
$\wedge\ NotRefresh_p(m)$
$\wedge\ k \in 1\,..\,Len(in[p])$
$\wedge\ NotRefresh_p(k)$
$\wedge\ k = m$
$\wedge\ in[p][k][1] = mem[in[p][m][2]]$
$\wedge\ in[p][k][2] = in[p][m][2]$
$\wedge\ \forall j \in k+1\,..\,Len(in[p]) :$
$\quad \vee\ in[p][j][2] \neq in[p][m][2]$
$\quad \vee\ in[p][j][1] = mem[in[p][m][2]]$
$\wedge\ \forall j \in m+1\,..\,Len(in[p]) : \neg NotRefresh_p(j)$
$\Rightarrow$
$\quad \forall j \in m+1\,..\,Len(in[p]) :$
$\quad\quad \vee\ in[p][j][2] \neq in[p][m][2]$
$\quad\quad \vee\ in[p][j][1] = in[p][m][1]$

PROOF: Follows immediately by substitution of $in[p][k][1]$ for $mem[in[p][m][2]]$ and m for k. □

Now the $k < m$ case.

$\langle 14\rangle 3.$ $\wedge\ i \in Proc \wedge HMemWrite(i)$
$\wedge\ p \in Proc$
$\wedge\ m \in 1\,..\,Len(in[p])$
$\wedge\ NotRefresh_p(m)$
$\wedge\ k \in 1\,..\,Len(in[p])$
$\wedge\ NotRefresh_p(k)$
$\wedge\ k < m$
$\wedge\ in[p][k][1] = mem[in[p][m][2]]$
$\wedge\ in[p][k][2] = in[p][m][2]$
$\wedge\ \forall j \in k+1\,..\,Len(in[p]) :$
$\quad \vee\ in[p][j][2] \neq in[p][m][2]$
$\quad \vee\ in[p][j][1] = mem[in[p][m][2]]$
$\wedge\ \forall j \in m+1\,..\,Len(in[p]) : \neg NotRefresh_p(j)$
$\Rightarrow$

$\forall j \in m+1 \,..\, Len(in[p])$:
$\quad \vee\ in[p][j][2] \neq in[p][m][2]$
$\quad \vee\ in[p][j][1] = in[p][m][1]$

PROOF:

$\langle 15 \rangle 1.$ $\wedge\ i \in Proc \wedge HMemWrite(i)$
$\wedge\ p \in Proc$
$\wedge\ m \in 1 \,..\, Len(in[p])$
$\wedge\ NotRefresh_p(m)$
$\wedge\ k \in 1 \,..\, Len(in[p])$
$\wedge\ NotRefresh_p(k)$
$\wedge\ k < m$
$\wedge\ in[p][k][1] = mem[in[p][m][2]]$
$\wedge\ in[p][k][2] = in[p][m][2]$
$\wedge\ \forall j \in k+1 \,..\, Len(in[p])$:
$\quad \vee\ in[p][j][2] \neq in[p][m][2]$
$\quad \vee\ in[p][j][1] = mem[in[p][m][2]]$
$\wedge\ \forall j \in m+1 \,..\, Len(in[p]) : \neg NotRefresh_p(j)$
$\Rightarrow in[p][m][1] = mem[in[p][m][2]]$

PROOF: It follows from hypothesis $k < m$ that $m \geq k+1$. Instantianting j to m in the quantifier $\forall j \in k+1 \,..\, Len(in[p])$ in the hypothesis yields the conclusion, since one disjunct, $in[p][m][2] \neq in[p][m][2]$, is a contradiction. □

$\langle 15 \rangle 2.$ $\wedge\ i \in Proc \wedge HMemWrite(i)$
$\wedge\ p \in Proc$
$\wedge\ m \in 1 \,..\, Len(in[p])$
$\wedge\ NotRefresh_p(m)$
$\wedge\ k \in 1 \,..\, Len(in[p])$
$\wedge\ NotRefresh_p(k)$
$\wedge\ k < m$
$\wedge\ in[p][k][1] = mem[in[p][m][2]]$
$\wedge\ in[p][k][2] = in[p][m][2]$
$\wedge\ \forall j \in k+1 \,..\, Len(in[p])$:
$\quad \vee\ in[p][j][2] \neq in[p][m][2]$
$\quad \vee\ in[p][j][1] = mem[in[p][m][2]]$
$\wedge\ \forall j \in m+1 \,..\, Len(in[p]) : \neg NotRefresh_p(j)$
$\Rightarrow$
$\quad \forall j \in m+1 \,..\, Len(in[p])$:
$\quad\quad \vee\ in[p][j][2] \neq in[p][m][2]$
$\quad\quad \vee\ in[p][j][1] = mem[in[p][m][2]]$

PROOF: The consequent follows immediately from the similar hypothesis by restricting the quantifier. □

$\langle 15 \rangle 3.$ Q.E.D.

PROOF: Follows from $\langle 15 \rangle 2$ by substituting $in[p][m][1]$ for $mem[in[p][m][2]]$ in the conclusion, according to the conclusion of $\langle 15 \rangle 1$. □

$\langle 14 \rangle 4.$ Q.E.D.

PROOF: Immediate from $\langle 14 \rangle 1$, $\langle 14 \rangle 2$, $\langle 14 \rangle 3$, and the truth $k < m \vee k = m \vee m < k$, which follows from axioms in the *Naturals* module. □

$\langle 13 \rangle 2.$ Q.E.D.

PROOF: Follows immediately from $\langle 13 \rangle 1$ by existential generalisation on k. □

$\langle 12 \rangle 3.$ Q.E.D.

PROOF: Follows directly by propositional logic from $\langle 12 \rangle 1$ and $\langle 12 \rangle 2$. □

$\langle 11 \rangle 2.$ Q.E.D.

PROOF: Follows by propositional logic from $\langle 11 \rangle 1$ since the disjunctive hypothesis in $\langle 10 \rangle 1$ is strengthened by the universal quantification. □

⟨10⟩2. Q.E.D.

PROOF: Immediate by propositional logic from ⟨10⟩1, by weakening the consequent. □

⟨9⟩4. Q.E.D.

PROOF: Follows from ⟨9⟩2 and ⟨9⟩3 using universal generalisation over m, n, and then by ⟨9⟩1. □

We now show that all the other components, *FirstIsVC*, *LastIsMem*, and *NoMemWrite*, are preserved.

We first show that *LastIsMem* is preserved for $a = \mathit{Head}(\mathit{out}[i])[2]$.

⟨8⟩2. $\land\ i \in \mathit{Proc} \land \mathit{HMemWrite}(i)$
$\land\ p \in \mathit{Proc}$
$\Rightarrow\quad \mathit{LastIsMem}'(\mathit{Head}(\mathit{out}[i])[2], p)$

PROOF:

⟨9⟩1. $\land\ i \in \mathit{Proc} \land \mathit{HMemWrite}(i)$
$\land\ p \in \mathit{Proc}$
$\Rightarrow$
$\land\ \mathit{NotRefresh}_p{}'(\mathit{Len}(\mathit{in}'[p])$
$\land\ \mathit{in}'[p][\mathit{Len}(\mathit{in}'[p])][1] = \mathit{mem}'[\mathit{Head}(\mathit{out}[i])[2]]$
$\land\ \mathit{in}'[p][\mathit{Len}(\mathit{in}'[p])][2] = \mathit{Head}(\mathit{out}[i])[2]$

PROOF:

⟨10⟩1. $\mathit{hin}'[p][\mathit{Len}(\mathit{hin}'[p])][2] =$ "Wr"

PROOF: Immediate from the definition of *HMemWrite*. □

⟨10⟩2. $i \in \mathit{Proc} \land \mathit{HMemWrite}(i) \Rightarrow \mathit{mem}'[\mathit{Head}(\mathit{out}[i])[2]] = \mathit{Head}(\mathit{out}[i])[1]$

PROOF: Immediate from the definition of *HMemWrite*. □

⟨10⟩3. Q.E.D.

PROOF: The first conjunct follows immediately from ⟨10⟩1, and the second and third from ⟨10⟩2 using the definition of *HMemWrite*. □

⟨9⟩2. Q.E.D.

PROOF: Immediate from the definition of *LastIsMem* and ⟨9⟩1. □

Second, *FirstIsVC* is preserved. This is split into two pieces. The hypotheses are used to prove the consequent conjuncts except for *NotRefresh*′; and then they are used to show *NotRefresh*′.

⟨8⟩3. $\land\ i \in \mathit{Proc}$
$\land\ \mathit{HMemWrite}(i)$
$\land\ p \in \mathit{Proc}$
$\land\ \mathit{CCInv}.1(p)$
$\land\ a \in \mathit{Addr}$
$\land\ \mathit{FirstIsVC}(a, p)$
$\Rightarrow \mathit{FirstIsVC}'(a, p)$

PROOF:

LET: k : CONSTANT

$\langle 9 \rangle 1.$ $\land\ i \in Proc$
$\land\ HMemWrite(i)$
$\land\ p \in Proc$
$\land\ CCInv.1(p)$
$\land\ a \in Addr$
$\land\ k \in 1\,..\,Len(in[p])$
$\land\ NotRefresh_p(k)$
$\land\ in[p][k][2] = a$
$\land\ \forall j \in 1\,..\,k-1 : \lor\ in[p][j][2] \neq a$
$\qquad\qquad\qquad\qquad \lor\ in[p][j][1] = vc[p][a]$
$\Rightarrow$
$\quad \land\ p \in Proc$
$\quad \land\ k \in 1\,..\,Len(in'[p])$
$\quad \land\ in'[p][k][2] = a$
$\quad \land\ \forall j \in 1\,..\,k-1 : \lor\ in'[p][j][2] \neq a$
$\qquad\qquad\qquad\qquad\quad \lor\ in'[p][j][1] = vc'[p][a]$

PROOF:

$\langle 10 \rangle 1.$ $\land\ i \in Proc \land HMemWrite(i)$
$\land\ p \in Proc$
$\land\ CCInv.1(p)$
$\land\ a \in Addr$
$\land\ k \in 1\,..\,Len(in[p])$
$\land\ NotRefresh_p(k)$
$\land\ in[p][k][2] = a$
$\land\ j \in 1\,..\,k-1$
$\land\ \quad \lor\ in[p][j][2] \neq a$
$\qquad \lor\ in[p][j][1] = vc[p][a]$
$\Rightarrow$
$\quad \land\ p \in Proc$
$\quad \land\ k \in 1\,..\,Len(in'[p])$
$\quad \land\ in'[p][k][2] = a$
$\quad \land\ j \in 1\,..\,k-1$
$\quad \land\ \lor\ in'[p][j][2] \neq a$
$\qquad \lor\ in'[p][j][1] = vc'[p][a]$

PROOF:

$\langle 11 \rangle 1.$ $\land\ i \in Proc$
$\land\ HMemWrite(i)$
$\land\ p \in Proc$
$\land\ k \in 1\,..\,Len(in[p])$
$\quad \Rightarrow k \in 1\,..\,Len(in'[p])$

PROOF: Immediate from $\langle 7 \rangle 1$ and elementary set theory. □

$\langle 11 \rangle 2.$ Q.E.D.

PROOF: Immediate from $\langle 7 \rangle 4$ and $\langle 11 \rangle 1$. □

$\langle 10 \rangle 2.$ Q.E.D.

PROOF: Follows by universal generalisation over j and predicate logic from $\langle 10 \rangle 1$. □

$\langle 9 \rangle 2.$ $\land\ i \in Proc \land HMemWrite(i)$
$\land\ p \in Proc$
$\land\ CCInv.1(p)$
$\land\ a \in Addr$
$\land\ k \in 1 \,..\, Len(in[p])$
$\land\ NotRefresh_p(k)$
$\land\ in[p][k][2] = a$
$\land\ \forall j \in 1 \,..\, k-1 : \lor\ in[p][j][2] \neq a$
$\qquad\qquad\qquad\qquad \lor\ in[p][j][1] = vc[p][a]$
$\Rightarrow NotRefresh_p{}'(k)$

PROOF:

$\langle 10 \rangle 1.$ $\land\ i \in Proc$
$\land\ HMemWrite(i)$
$\land\ p \in Proc$
$\land\ k \in 1 \,..\, Len(hin[p])$
$\quad \Rightarrow k \in 1 \,..\, Len(hin'[p])$

PROOF: Immediate from $\langle 7 \rangle 1$ and elementary set theory. $\square$

$\langle 10 \rangle 2.$ $\land\ i \in Proc$
$\land\ HMemWrite(i)$
$\land\ p \in Proc$
$\land\ NotRefresh_p(k)$
$\quad \Rightarrow k \in 1 \,..\, Len(hin[p])$

PROOF: Immediate from the definition of *NotRefresh*. $\square$

$\langle 10 \rangle 3.$ Q.E.D.

Immediate by propositional logic from $\langle 10 \rangle 1$ and $\langle 10 \rangle 2$, and then using $\langle 7 \rangle 4$. $\square$

$\langle 9 \rangle 3.$ Q.E.D.

PROOF: Immediate, by propositional logic from $\langle 9 \rangle 1$ and $\langle 9 \rangle 2$, and then existential generalisation over k. $\square$

Third, *NoMemWrite* is preserved, for all $a \in Addr$ except for $a = Head(out[i])[2]$.

$\langle 8 \rangle 4.$ $\land\ i \in Proc \land HMemWrite(i)$
$\land\ p \in Proc$
$\land\ a \in Addr \setminus \{Head(out[i])[2]\}$
$\land\ NoMemWrite(a, p)$
$\quad \Rightarrow NoMemWrite'(a, p)$

PROOF:

The substep is simply a substitution of the definition of *HMemWrite*.

$\langle 9 \rangle 1.$ $\land\ i \in Proc \land HMemWrite(i)$
$\land\ p \in Proc$
$\land\ a \in Addr \setminus \{Head(out[i])[2]\}$
$\land\ vc[p][a] = mem[a]$
$\land\ \forall k \in 1 \,..\, Len(in[p]) : \lor\ in[p][k][2] \neq a$
$\qquad\qquad\qquad\qquad\qquad \lor\ in[p][k][1] = mem[a]$
$\Rightarrow$
$\quad \land\ vc'[p][a] = mem'[a]$
$\quad \land\ \forall k \in 1 \,..\, Len(in'[p]) : \lor\ in'[p][k][2] \neq a$
$\qquad\qquad\qquad\qquad\qquad\quad \lor\ in'[p][k][1] = mem'[a]$

PROOF:

$\langle 10\rangle 1.\ \land\ i \in Proc \land HMemWrite(i)$
$\quad\land\ p \in Proc$
$\quad\land\ a \in Addr \setminus \{Head(out[i])[2]\}$
$\quad\land\ k \in 1\,..\,Len(in[p])$
$\quad\land\ \lor\ in[p][k][2] \neq a$
$\qquad\lor\ in[p][k][1] = mem[a]$
$\quad\Rightarrow\ \land\ k \in 1\,..\,Len(in'[p])$
$\qquad\land\ \lor\ in'[p][k][2] \neq a$
$\qquad\quad\lor\ in'[p][k][1] = mem'[a]$

Proof:

We split the proof into two: $k \in 1\,..\,Len(in[p])$ and $k = Len(in'[p])$. First, $k \in 1\,..\,Len(in[p])$.

$\langle 11\rangle 1.\ \land\ i \in Proc \land HMemWrite(i)$
$\quad\land\ p \in Proc$
$\quad\land\ a \in Addr \setminus \{Head(out[i])[2]\}$
$\quad\land\ k \in 1\,..\,Len(in[p])$
$\quad\land\ \lor\ in[p][k][2] \neq a$
$\qquad\lor\ in[p][k][1] = mem[a]$
$\quad\Rightarrow\ \land\ k \in 1\,..\,Len(in[p])\ :$
$\qquad\land\ \lor\ in'[p][k][2] \neq a$
$\qquad\quad\lor\ in'[p][k][1] = mem'[a]$

Proof: Immediate from $\langle 7\rangle 5$, $\langle 7\rangle 4$, and propositional logic. □

Next, $k = Len(in'[p])$.

$\langle 11\rangle 2.\ \land\ i \in Proc \land HMemWrite(i)$
$\quad\land\ p \in Proc$
$\quad\land\ a \in Addr \setminus \{Head(out[i])[2]\}$
$\quad\Rightarrow\ in'[p][Len(in'[p])][2] \neq a$

Proof:

$\langle 12\rangle 1.\ \land\ i \in Proc \land HMemWrite(i)$
$\quad\land\ p \in Proc$
$\quad\Rightarrow\ in'[p][Len(in'[p])][2] = Head(out[i])[2]$

Proof: Immediate from the definition of $HMemWrite$. □

$\langle 12\rangle 2.$ Q.E.D.

Proof: Immediate by equality reasoning and set theory definitions from $\langle 12\rangle 1$. □

$\langle 11\rangle 3.$ Q.E.D.

Proof: Follows from $\langle 11\rangle 1$ and $\langle 11\rangle 2$ by $\langle 7\rangle 3$ and propositional logic. □

$\langle 10\rangle 2.$ Q.E.D.

Proof: Immediate from $\langle 10\rangle 1$ by universal generalisation over k and predicate logic, and by adding the premis $vc[p][a] = mem[a]$ and conclusion $vc'[p][a] = mem'[a]$ and using $\langle 7\rangle 4$ and $\langle 7\rangle 5$. □

$\langle 9\rangle 2.$ Q.E.D.

Proof: Immediate from $\langle 9\rangle 1$ by substitution for the definition of $NoMemWrite$. □

Fourth, $NoMemWrite$ is not quite preserved for $a = Head(out[i])[2]$.

$\langle 8\rangle 5.\ \land\ i \in Proc \land HMemWrite(i)$
$\quad\land\ p \in Proc$
$\quad\land\ CCInv.1(p)$
$\quad\land\ NoMemWrite(Head(out[i])[2], p)$

$$\begin{array}{l}
\Rightarrow \lor\ \mathit{NoMemWrite}'(\mathit{Head}(\mathit{out}[i])[2], p) \\
\quad \lor\ \land\ \mathit{LastIsMem}'(\mathit{Head}(\mathit{out}[i])[2], p) \\
\quad\quad \land\ \mathit{FirstIsVC}'(\mathit{Head}(\mathit{out}[i])[2], p)
\end{array}$$

PROOF:

$\langle 9 \rangle 1.$
$$\begin{array}{l}
\land\ i \in \mathit{Proc} \land \mathit{HMemWrite}(i) \\
\land\ p \in \mathit{Proc} \\
\land\ \mathit{CCInv}.1(p) \\
\land\ \mathit{NoMemWrite}(\mathit{Head}(\mathit{out}[i])[2], p) \\
\Rightarrow \lor\ \mathit{NoMemWrite}'(\mathit{Head}(\mathit{out}[i])[2], p) \\
\quad \lor\ \mathit{FirstIsVC}'(\mathit{Head}(\mathit{out}[i])[2], p)
\end{array}$$

PROOF:

$\langle 10 \rangle 1.$
$$\begin{array}{l}
\land\ i \in \mathit{Proc} \land \mathit{HMemWrite}(i) \\
\land\ p \in \mathit{Proc} \\
\land\ \mathit{CCInv}.1(p) \\
\land\ \mathit{vc}[p][\mathit{Head}(\mathit{out}[i])[2]] = \mathit{mem}[\mathit{Head}(\mathit{out}[i])[2]] \\
\land\ \forall k \in 1 \,..\, \mathit{Len}(\mathit{in}[p]) : \begin{array}[t]{l} \lor\ \mathit{in}[p][k][2] \neq \mathit{Head}(\mathit{out}[i])[2] \\ \lor\ \mathit{in}[p][k][1] = \mathit{mem}[\mathit{Head}(\mathit{out}[i])[2]] \end{array} \\
\Rightarrow \lor\ \land\ p \in \mathit{Proc} \\
\quad\quad \land\ \mathit{vc}'[p][\mathit{Head}(\mathit{out}[i])[2]] = \mathit{mem}'[\mathit{Head}(\mathit{out}[i])[2]] \\
\quad\quad \land\ \forall k \in 1 \,..\, \mathit{Len}(\mathit{in}'[p]) : \begin{array}[t]{l} \lor\ \mathit{in}'[p][k][2] \neq \mathit{Head}(\mathit{out}[i])[2] \\ \lor\ \mathit{in}'[p][k][1] = \mathit{mem}'[\mathit{Head}(\mathit{out}[i])[2]] \end{array} \\
\quad \lor\ \land\ p \in \mathit{Proc} \\
\quad\quad \land\ \exists k \in 1 \,..\, \mathit{Len}(\mathit{in}'[p]) : \\
\quad\quad\quad \land\ \mathit{NotRefresh}'_p(k) \\
\quad\quad\quad \land\ \mathit{in}'[p][k][2] = \mathit{Head}(\mathit{out}'[i])[2] \\
\quad\quad\quad \land\ \forall j \in 1 \,..\, k-1 : \begin{array}[t]{l} \lor\ \mathit{in}'[p][j][2] \neq \mathit{Head}(\mathit{out}'[i])[2] \\ \lor\ \mathit{in}'[p][j][1] = \mathit{vc}'[p][\mathit{Head}(\mathit{out}'[i])[2]] \end{array}
\end{array}$$

PROOF:

We split into two cases, first $\mathit{mem}[\mathit{Head}(\mathit{out}[i])[2]] = \mathit{mem}'[\mathit{Head}(\mathit{out}[i])[2]]$ and second $\mathit{mem}[\mathit{Head}(\mathit{out}[i])[2]] \neq \mathit{mem}'[\mathit{Head}(\mathit{out}[i])[2]]$.

$\langle 11 \rangle 1.$
$$\begin{array}{l}
\land\ i \in \mathit{Proc} \land \mathit{HMemWrite}(i) \\
\land\ p \in \mathit{Proc} \\
\land\ \mathit{CCInv}.1(p) \\
\land\ \mathit{vc}[p][\mathit{Head}(\mathit{out}[i])[2]] = \mathit{mem}[\mathit{Head}(\mathit{out}[i])[2]] \\
\land\ \mathit{mem}[\mathit{Head}(\mathit{out}[i])[2]] = \mathit{mem}'[\mathit{Head}(\mathit{out}[i])[2]] \\
\land\ \forall k \in 1 \,..\, \mathit{Len}(\mathit{in}[p]) : \begin{array}[t]{l} \lor\ \mathit{in}[p][k][2] \neq \mathit{Head}(\mathit{out}[i])[2] \\ \lor\ \mathit{in}[p][k][1] = \mathit{mem}[\mathit{Head}(\mathit{out}[i])[2]] \end{array} \\
\Rightarrow \land\ p \in \mathit{Proc} \\
\quad \land\ \mathit{vc}'[p][\mathit{Head}(\mathit{out}[i])[2]] = \mathit{mem}'[\mathit{Head}(\mathit{out}[i])[2]] \\
\quad \land\ \forall k \in 1 \,..\, \mathit{Len}(\mathit{in}'[p]) : \begin{array}[t]{l} \lor\ \mathit{in}'[p][k][2] \neq \mathit{Head}(\mathit{out}[i])[2] \\ \lor\ \mathit{in}'[p][k][1] = \mathit{mem}'[\mathit{Head}(\mathit{out}[i])[2]] \end{array}
\end{array}$$

PROOF: Follows directly by substitution of $\mathit{mem}[\mathit{Head}(\mathit{out}[i])[2]]$ for $\mathit{mem}'[\mathit{Head}(\mathit{out}[i])[2]]$, of $\mathit{vc}[p][\mathit{Head}(\mathit{out}[i])[2]]$ for $\mathit{vc}'[p][\mathit{Head}(\mathit{out}[i])[2]]$ by $\langle 7 \rangle 4$, and using $\langle 6 \rangle 2$ on the set $1 \,..\, \mathit{Len}(\mathit{in}'[p])$ by $\langle 7 \rangle 3$. □

$\langle 11 \rangle 2.$
$$\begin{array}{l}
\land\ i \in \mathit{Proc} \land \mathit{HMemWrite}(i) \\
\land\ p \in \mathit{Proc} \\
\land\ \mathit{CCInv}.1(p) \\
\land\ \mathit{vc}[p][\mathit{Head}(\mathit{out}[i])[2]] = \mathit{mem}[\mathit{Head}(\mathit{out}[i])[2]] \\
\land\ \mathit{mem}[\mathit{Head}(\mathit{out}[i])[2]] \neq \mathit{mem}'[\mathit{Head}(\mathit{out}[i])[2]] \\
\land\ \forall k \in 1 \,..\, \mathit{Len}(\mathit{in}[p]) : \begin{array}[t]{l} \lor\ \mathit{in}[p][k][2] \neq \mathit{Head}(\mathit{out}[i])[2] \\ \lor\ \mathit{in}[p][k][1] = \mathit{mem}[\mathit{Head}(\mathit{out}[i])[2]] \end{array}
\end{array}$$

$$\begin{array}{l}
\Rightarrow \land\ p \in Proc \\
\quad \land\ \exists\, k \in 1\,..\,Len(in'[p]) \ : \\
\quad\quad \land\ NotRefresh'_p(k) \\
\quad\quad \land\ in'[p][k][2] = Head(out'[i])[2] \\
\quad\quad \land\ \forall\, j \in 1\,..\,k-1 \ : \lor\ in'[p][j][2] \neq Head(out[i])[2] \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \lor\ in'[p][j][1] = vc'[p][Head(out[i])[2]]
\end{array}$$

PROOF:

$\langle 12\rangle 1.$ $\land\ i \in Proc \land HMemWrite(i)$

$$\begin{array}{l}
\land\ p \in Proc \\
\land\ CCInv.1(p) \\
\land\ vc[p][Head(out[i])[2]] = mem[Head(out[i])[2]] \\
\land\ mem[Head(out[i])[2]] \neq mem'[Head(out[i])[2]] \\
\land\ \forall\, k \in 1\,..\,Len(in[p]) \ : \lor\ in[p][k][2] \neq Head(out[i])[2] \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \lor\ in[p][k][1] = mem[Head(out[i])[2]] \\
\Rightarrow \land\ p \in Proc \\
\quad \land\ NotRefresh'_p(Len(in'[p])) \\
\quad \land\ in'[p][Len(in'[p])][2] = Head(out[i])[2] \\
\quad \land\ \forall\, j \in 1\,..\,Len(in[p]) \lor\ in'[p][j][2] \neq Head(out[i])[2] \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \lor\ in'[p][j][1] = vc'[p][Head(out[i])[2]]
\end{array}$$

PROOF: The first conjunct of the consequent is trivial, the second follows from the definition of *NotRefresh* and *HMemWrite*, the third by definition of *HMemWrite* and $Len(in'[p])$, and the last by the last hypothesis, noting that $vc'[p][Head(out[i])[2]] = vc[p][Head(out[i])[2]]$, and that $\forall\, j \in 1\,..\,Len(in[p]) \ : in'[p][j] = in[p][j]$, both by $\langle 7\rangle 4$ and substituting. □

$\langle 12\rangle 2.$ Q.E.D.

PROOF: Follows immediately from $\langle 12\rangle 1$ by substitution of $k-1$ for $Len(in[p])$ by $\langle 7\rangle 1$, by existential generalisation on k, and propositional logic. □

$\langle 11\rangle 3.$ Q.E.D.

PROOF: By propositional logic from $\langle 11\rangle 1$ and $\langle 11\rangle 1$. □

$\langle 10\rangle 2.$ Q.E.D.

PROOF: $\langle 10\rangle 1$ is simply a rewrite of $\langle 9\rangle 1$. □

$\langle 9\rangle 2.$ Q.E.D.

PROOF: Follows by propositional logic from $\langle 9\rangle 1$ and $\langle 8\rangle 2$, which says that *LastIsMem* is preserved for this value of a. □

Fifth, *LastIsMem* is preserved.

$\langle 8\rangle 6.$ $\land\ i \in Proc \land HMemWrite(i)$

$$\begin{array}{l}
\land\ p \in Proc \\
\land\ LastIsMem(a, p) \\
\quad \Rightarrow LastIsMem'(a, p)
\end{array}$$

PROOF:

Since this follows directly from $\langle 8\rangle 2$ for $a = Head(out[i])[2]$, we need to show it for the other values of a.

$\langle 9\rangle 1.$ $\land\ i \in Proc \land HMemWrite(i)$

$$\begin{array}{l}
\land\ p \in Proc \\
\land\ a \in Addr \setminus \{Head(out[i])[2]\} \\
\land\ LastIsMem(a, p) \\
\quad \Rightarrow LastIsMem'(a, p)
\end{array}$$

PROOF:

The substep is simply to expand the definition of *LastIsMem*.

$\langle 10\rangle 1.$ $\land\ i \in Proc \land HMemWrite(i)$
$\land\ p \in Proc$
$\land\ a \in Addr \setminus \{Head(out[i])[2]\}$
$\land\ \exists\, k \in 1 \,..\, Len(in[p])\ :$
$\quad \land\ NotRefresh_p(k)$
$\quad \land\ in[p][k][1] = mem[a]$
$\quad \land\ in[p][k][2] = a$
$\quad \land\ \forall\, j \in k+1 \,..\, Len(in[p])\ :\ \lor\ in[p][j][2] \neq a$
$\qquad\qquad\qquad\qquad\qquad\qquad \lor\ in[p][j][1] = mem[a]$
$\Rightarrow$
$\quad \exists\, k \in 1 \,..\, Len(in'[p])\ :$
$\qquad \land\ NotRefresh_p{}'(k)$
$\qquad \land\ in'[p][k][1] = mem'[a]$
$\qquad \land\ in'[p][k][2] = a$
$\qquad \land\ \forall\, j \in k+1 \,..\, Len(in'[p])\ :\ \lor\ in'[p][j][2] \neq a$
$\qquad\qquad\qquad\qquad\qquad\qquad \lor\ in'[p][j][1] = mem'[a]$

PROOF:

The substep is to instantiate k. The same k will witness both hypothesis and consequent, since it can't be $Len(in'[p])$ in the consequent by the hypothesis on the value of a.

LET: k : CONSTANT

$\langle 11\rangle 1.$ $\land\ i \in Proc \land HMemWrite(i)$
$\land\ p \in Proc$
$\land\ a \in Addr \setminus \{Head(out[i])[2]\}$
$\land\ k \in 1 \,..\, Len(in[p])$
$\land\ NotRefresh_p(k)$
$\land\ in[p][k][1] = mem[a]$
$\land\ in[p][k][2] = a$
$\land\ \forall\, j \in k+1 \,..\, Len(in[p])\ :\ \lor\ in[p][j][2] \neq a$
$\qquad\qquad\qquad\qquad\qquad\qquad \lor\ in[p][j][1] = mem[a]$
$\Rightarrow$
$\quad \land\ k \in 1 \,..\, Len(in'[p])$
$\quad \land\ NotRefresh_p{}'(k)$
$\quad \land\ in'[p][k][1] = mem'[a]$
$\quad \land\ in'[p][k][2] = a$
$\quad \land\ \forall\, j \in k+1 \,..\, Len(in[p])\ :\ \lor\ in'[p][j][2] \neq a$
$\qquad\qquad\qquad\qquad\qquad\qquad \lor\ in'[p][j][1] = mem'[a]$
$\quad \land\ in'[p][Len(in'[p])][2] \neq a$

PROOF: Immediate from $\langle 7\rangle 1$, $\langle 7\rangle 5$ and $\langle 7\rangle 4$. □

$\langle 11\rangle 2.$ Q.E.D.

PROOF: Immediate by predicate logic from $\langle 11\rangle 1$ by using $\langle 7\rangle 3$ and $\langle 6\rangle 2$ to combine the last two conjuncts of the consequent, and by existentially quantifying over k in antecedent and consequent and predicate logic. □

$\langle 10\rangle 2.$ Q.E.D.

PROOF: Immediate from $\langle 10\rangle 1$ by definition of *LastIsMem*. □

$\langle 9\rangle 2.$ Q.E.D.

PROOF: Immediate from $\langle 8\rangle 2$ and $\langle 9\rangle 1$ using $\langle 6\rangle 2$. □

That completes the sequence of steps showing the preservation of components of $CCInv.3(p)$ under *HMemWrite*. We conclude thus:-

$\langle 8 \rangle$7. Q.E.D.

PROOF: Follows by propositional logic using $\langle 8 \rangle$1, $\langle 8 \rangle$3,$\langle 8 \rangle$6,and using $\langle 6 \rangle$2 on $\langle 8 \rangle$4 and $\langle 8 \rangle$5. $\square$

We may now conclude that all components of *CCInv* are preserved under *HMemWrite*:-

$\langle 7 \rangle$10. Q.E.D.

PROOF: Follows by propositional logic and quantifier rules from $\langle 7 \rangle$7, $\langle 7 \rangle$8, and $\langle 7 \rangle$9. $\square$

The invariant is preserved by a *HCacheUpdate*

$\langle 6\rangle 6.$ $CCInv \wedge i \in Proc \wedge HCacheUpdate(i) \Rightarrow CCInv'$
PROOF:

First, two logical properties of sequence operations and a general logical property.

LET: k : CONSTANT
$\langle 7\rangle 1.$ $k \in 2 \,..\, Len(s) \Rightarrow \phi(s[k]) \equiv \phi(Tail(s)[k-1])$
PROOF: Logical property of *Tail*, assumed without further proof. $\square$
$\langle 7\rangle 2.$ $\forall\, k \in 2 \,..\, Len(s) \;:\; \phi(s[k]) \equiv \forall\, p \in 1 \,..\, Len(Tail(s)) \;:\; \phi(Tail(s)[p])$
PROOF: Follows from $\langle 7\rangle 1$ by properties of the sets $q \,..\, r$ and universal generalisation. $\square$
$\langle 7\rangle 3.$ $\forall\, x \in S \;:\; \phi \wedge T \subset S \Rightarrow \forall\, x \in T \;:\; \phi$
PROOF: This is general logic and set theory, which we assume without further proof. $\square$

Some general properties of *HCacheUpdate*.

$\langle 7\rangle 4.$ $i \in Proc \wedge HCacheUpdate(i) \Rightarrow \wedge\; Len(in'[i]) = Len(in[i]) - 1$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge\; Len(hin'[i]) = Len(hin[i]) - 1$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge\; \text{UNCHANGED}\; \langle out, hout\rangle$
PROOF: Immediate from the definitions of *HCacheUpdate* and *Tail*. $\square$
$\langle 7\rangle 5.$ $\wedge\; i \in Proc$
$\wedge\; HCacheUpdate(i)$
$\wedge\; (k \in 2 \,..\, Len(in[i]))$
$\Rightarrow \wedge\; NotRefresh_i(k) \equiv NotRefresh_i{}'(k-1)$
$\quad \wedge\; in[i][k] = in'[i][k-1]$
PROOF: Follows from the definition of *HCacheUpdate*, and $\langle 7\rangle 1$. $\square$

$CCInv(i)$ is preserved under $HCacheUpdate(i)$.

$\langle 7\rangle 6.$ $\wedge\; i \in Proc$
$\wedge\; CCInv(i)$
$\wedge\; HCacheUpdate(i)$
$\Rightarrow CCInv'(i)$
PROOF:

First, $CCInv.1(i)$ is preserved under $HCacheUpdate(i)$.

$\langle 8\rangle 1.$ $\wedge\; i \in Proc$
$\wedge\; CCInv.1(i)$
$\wedge\; HCacheUpdate(i)$
$\Rightarrow CCInv.1'(i)$
PROOF: Follows immediately from the definition of *HCacheUpdate* by $\langle 7\rangle 4$ for the first two conjuncts of *CCInv*.1, and by $\langle 7\rangle 3$ for the third conjunct. $\square$

$CCInv.2(i)$ is preserved under $HCacheUpdate(i)$.

$\langle 8\rangle 2.$ $\wedge\; i \in Proc$
$\wedge\; CCInv(i)$
$\wedge\; HCacheUpdate(i)$
$\Rightarrow CCInv.2'(i)$
PROOF: Follows immediately from logical equality rules since the new values of c and vc are defined by the same terms in the definition of *HCacheUpdate*. $\square$

$CCInv.3(i)$ is preserved under $HCacheUpdate(i)$.

$\langle 8 \rangle 3.$ $\land\ i \in Proc$
$\land\ CCInv(i)$
$\land\ HCacheUpdate(i)$
$\Rightarrow\ CCInv.3'(i)$

PROOF:

We show that each of the components of $CCInv.3(i)$ is preserved. First, $BetweenMemWrites$.

$\langle 9 \rangle 1.$ $\land\ i \in Proc$
$\land\ HCacheUpdate(i)$
$\land\ BetweenMemWrites(i)$
$\Rightarrow\ BetweenMemWrites'(i)$

PROOF: Immediate from restricting the universal quantifiers in the definition of $BetweenMemWrites$ and using $\langle 7 \rangle 2$. □

Next, $NoMemWrite$ is preserved.

$\langle 9 \rangle 2.$ $\land\ i \in Proc$
$\land\ a \in Addr$
$\land\ HCacheUpdate(i)$
$\land\ NoMemWrite(a, i)$
$\Rightarrow\ NoMemWrite'(a, i)$

PROOF:

This is split into two steps: $a = Head(in[i])[2]$ and $a \in Addr \setminus \{Head(in[i])[2]\}$

$\langle 10 \rangle 1.$ $\land\ i \in Proc$
$\land\ a \in Addr$
$\land\ HCacheUpdate(i)$
$\land\ NoMemWrite(Head(in[i])[2], i)$
$\Rightarrow\ NoMemWrite'(Head(in[i])[2], i)$

PROOF:

We expand the definition of $NoMemWrite$.

$\langle 11 \rangle 1.$
$$\begin{array}{l}
\land\ i \in Proc \land HCacheUpdate(i) \\
\land\ a \in Addr \\
\land\ vc[i][Head(in[i])[2]] = mem[Head(in[i])[2]] \\
\land\ \forall\, k \in 1\,..\,Len(in[i]) : \lor\ in[i][k][2] \neq Head(in[i])[2] \\
\qquad\qquad\qquad\qquad\qquad\quad \lor\ in[i][k][1] = mem[Head(in[i])[2]] \\
\Rightarrow\ \land\ vc'[i][Head(in[i])[2]] = mem'[Head(in[i])[2]] \\
\qquad \land\ \forall\, k \in 1\,..\,Len(in'[i]) : \lor\ in'[i][k][2] \neq Head(in[i])[2] \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \lor\ in'[i][k][1] = mem'[Head(in[i])[2]]
\end{array}$$

PROOF:

$\langle 12 \rangle 1.$
$$\begin{array}{l}
\land\ i \in Proc \land HCacheUpdate(i) \\
\land\ a \in Addr \\
\land\ vc[i][Head(in[i])[2]] = mem[Head(in[i])[2]] \\
\land\ in[i][1][1] = mem[Head(in[i])[2]] \\
\land\ \forall\, k \in 2\,..\,Len(in[i]) : \lor\ in[i][k][2] \neq Head(in[i])[2] \\
\qquad\qquad\qquad\qquad\qquad\quad \lor\ in[i][k][1] = mem[Head(in[i])[2]]
\end{array}$$

$$\Rightarrow \land\ vc'[i][Head(in[i])[2]] = mem'[Head(in[i])[2]]$$
$$\land\ \forall\, k \in 1\,..\,Len(in'[i]) \ :\ \lor\ in'[i][k][2] \neq Head(in[i])[2]$$
$$\lor\ in'[i][k][1] = mem'[Head(in[i])[2]]$$

Proof: First, substitute *mem* for mem', since *mem* is unchanged by definition of *HCacheUpdate*. The first clause of the consequent follows by substituting $mem[Head(in[i])[2]]$ for $Head(in[i])[1] = in[i][1][1]$ in the definition of *HCacheUpdate*. The second clause follows from the corresponding clause of the antecedent by $\langle 7\rangle 2$. □

$\langle 12\rangle 2$. Q.E.D.

Proof: Follows from $\langle 12\rangle 1$ using $\langle 6\rangle 2$ on the last two clauses of the antecedent, and the definition of $s\,..\,t$. □

$\langle 11\rangle 2$. Q.E.D.

Proof: $\langle 11\rangle 1$ is a rewrite of $\langle 10\rangle 1$ using the definition of *NoMemWrite*. □

Next, *NoMemWrite* is preserved for $a \in Addr \setminus \{Head(in[i])[2]\}$.

$\langle 10\rangle 2$. $\land\ i \in Proc \land HCacheUpdate(i)$
$\land\ a \in Addr \setminus \{Head(in[i])[2]\}$
$\land\ NoMemWrite(a, i)$
$\Rightarrow NoMemWrite'(a, i)$

Proof:

We expand the definition of *NoMemWrite*.

$\langle 11\rangle 1$. $\land\ i \in Proc \land HCacheUpdate(i)$
$\land\ a \in Addr \setminus \{Head(in[i])[2]\}$
$\land\ vc[i][a] = mem[a]$
$\land\ \forall\, k \in 1\,..\,Len(in[i]) \ :\ \lor\ in[i][k][2] \neq a$
$\lor\ in[i][k][1] = mem[a]$
$\Rightarrow$
$\land\ vc'[i][a] = mem'[a]$
$\land\ \forall\, k \in 1\,..\,Len(in'[i]) \ :\ \lor\ in'[i][k][2] \neq a$
$\lor\ in'[i][k][1] mem'[a]$

Proof:

We weaken the hypothesis quantifier $\forall\, k \in 1\,..\,Len(in'[i])$ to $\forall\, k \in 2\,..\,Len(in'[i])$, and use $\langle 7\rangle 2$ to obtain the conclusion. $vc[i]$ and *mem* are unchanged for this address value.

$\langle 12\rangle 1$. $\land\ i \in Proc \land HCacheUpdate(i)$
$\land\ a \in Addr \setminus \{Head(in[i])[2]\}$
$\land\ vc[i][a] = mem[a]$
$\land\ \forall\, k \in 2\,..\,Len(in[i]) \ :\ \lor\ in[i][k][2] \neq a$
$\lor\ in[i][k][1] = mem[a]$
$\Rightarrow$
$\land\ vc'[i][a] = mem'[a]$
$\land\ \forall\, k \in 1\,..\,Len(in'[i]) \ :\ \lor\ in'[i][k][2] \neq a$
$\lor\ in'[i][k][1] mem'[a]$

Proof: First, substitute $vc[i][a]$ for $vc'[i][a]$ and $mem[a]$ for $mem'[a]$ since both are unchanged by definition of *HCacheUpdate* and $a \neq Head(in[i])[2]$. The first conjunct of the consequent now follows from the identical conjunct of the antecedent. The second conjunct of the consequent follows from the final hypothesis by $\langle 7\rangle 2$. □

$\langle 12\rangle 2$. Q.E.D.

PROOF: Immediate from $\langle 12 \rangle 1$ by strengthening the quantified hypothesis using the definition of $s \, .. \, t$. $\square$

$\langle 11 \rangle 2$. Q.E.D.

PROOF: Immediate from $\langle 11 \rangle 1$ by substituting the definition of *NoMemWrite*. $\square$

$\langle 10 \rangle 3$. Q.E.D.

PROOF: Immediate from $\langle 10 \rangle 1$ and $\langle 10 \rangle 2$ using $\langle 6 \rangle 2$. $\square$

Next, *LastIsMem* is preserved.

$$\begin{array}{ll} \langle 9 \rangle 3. & \wedge\ i \in \mathit{Proc} \wedge \mathit{HCacheUpdate}(i) \\ & \wedge\ a \in \mathit{Addr} \\ & \wedge\ \mathit{LastIsMem}(a, i) \\ & \qquad \Rightarrow \mathit{LastIsMem}'(a, i) \end{array}$$

PROOF:

This is split into two steps: $a = \mathit{Head}(\mathit{in}[i])[2]$ and $a \in \mathit{Addr} \setminus \{\mathit{Head}(\mathit{in}[i])[2]\}$. First, $a \in \mathit{Addr} \setminus \{\mathit{Head}(\mathit{in}[i])[2]\}$.

$$\begin{array}{ll} \langle 10 \rangle 1. & \wedge\ i \in \mathit{Proc} \wedge \mathit{HCacheUpdate}(i) \\ & \wedge\ a \in \mathit{Addr} \setminus \{\mathit{Head}(\mathit{in}[i])[2]\} \\ & \wedge\ \mathit{LastIsMem}(a, i) \\ & \qquad \Rightarrow \mathit{LastIsMem}'(a, i) \end{array}$$

PROOF:

We expand the definition of *LastIsMem*.

$$\begin{array}{ll} \langle 11 \rangle 1. & \wedge\ i \in \mathit{Proc} \wedge \mathit{HCacheUpdate}(i) \\ & \wedge\ a \in \mathit{Addr} \setminus \{\mathit{Head}(\mathit{in}[i])[2]\} \\ & \wedge\ \exists\, k \in 1 \, .. \, \mathit{Len}(\mathit{in}[i]) \ : \\ & \qquad \wedge\ \mathit{NotRefresh}_i(k) \\ & \qquad \wedge\ \mathit{in}[i][k][1] = \mathit{mem}[a] \\ & \qquad \wedge\ \mathit{in}[i][k][2] = a \\ & \qquad \wedge\ \forall\, j \in k+1 \, .. \, \mathit{Len}(\mathit{in}[i]) \ : \ \vee\ \mathit{in}[i][j][2] \neq a \\ & \qquad \phantom{\wedge\ \forall\, j \in k+1 \, .. \, \mathit{Len}(\mathit{in}[i]) \ : \ } \vee\ \mathit{in}[i][j][1] = \mathit{mem}[a] \\ & \Rightarrow \exists\, k \in 1 \, .. \, \mathit{Len}(\mathit{in}'[i]) \ : \\ & \qquad \wedge\ \mathit{NotRefresh}_i{}'(k) \\ & \qquad \wedge\ \mathit{in}'[i][k][1] = \mathit{mem}'[a] \\ & \qquad \wedge\ \mathit{in}'[i][k][2] = a \\ & \qquad \wedge\ \forall\, j \in k+1 \, .. \, \mathit{Len}(\mathit{in}'[i]) \ : \ \vee\ \mathit{in}'[i][j][2] \neq a \\ & \qquad \phantom{\wedge\ \forall\, j \in k+1 \, .. \, \mathit{Len}(\mathit{in}'[i]) \ : \ } \vee\ \mathit{in}'[i][j][1] = \mathit{mem}'[a] \end{array}$$

PROOF:

We split the quantifier in the antecedent into $k = 1$ and $k \in 2 \, .. \, \mathit{Len}(\mathit{in}[i])$.

$$
\begin{array}{l}
\langle 12\rangle 1.\ \land\ i \in \mathit{Proc} \land \mathit{HCacheUpdate}(i) \\
\qquad \land\ a \in \mathit{Addr} \setminus \{\mathit{Head}(\mathit{in}[i])[2]\} \\
\qquad \land\ \lor\ \land\ \mathit{NotRefresh}_i(1) \\
\qquad\qquad\quad \land\ \mathit{in}[i][1][1] = \mathit{mem}[a] \\
\qquad\qquad\quad \land\ \mathit{in}[i][1][2] = a \\
\qquad\qquad\quad \land\ \forall j \in 2 \,..\, \mathit{Len}(\mathit{in}[i]) \ :\ \lor\ \mathit{in}[i][j][2] \neq a \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \lor\ \mathit{in}[i][j][1] = \mathit{mem}[a] \\
\qquad\quad\ \lor\ \exists k \in 2 \,..\, \mathit{Len}(\mathit{in}[i]) \ : \\
\qquad\qquad\quad \land\ \mathit{NotRefresh}_i(k) \\
\qquad\qquad\quad \land\ \mathit{in}[i][k][1] = \mathit{mem}[a] \\
\qquad\qquad\quad \land\ \mathit{in}[i][k][2] = a \\
\qquad\qquad\quad \land\ \forall j \in k+1 \,..\, \mathit{Len}(\mathit{in}[i]) \ :\ \lor\ \mathit{in}[i][j][2] \neq a \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \lor\ \mathit{in}[i][j][1] = \mathit{mem}[a] \\
\qquad \Rightarrow \exists k \in 1 \,..\, \mathit{Len}(\mathit{in}'[i]) \ : \\
\qquad\qquad \land\ \mathit{NotRefresh}_i{}'(k) \\
\qquad\qquad \land\ \mathit{in}'[i][k][1] = \mathit{mem}'[a] \\
\qquad\qquad \land\ \mathit{in}'[i][k][2] = a \\
\qquad\qquad \land\ \forall j \in k+1 \,..\, \mathit{Len}(\mathit{in}'[i]) \ :\ \lor\ \mathit{in}'[i][j][2] \neq a \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \lor\ \mathit{in}'[i][j][1] = \mathit{mem}'[a]
\end{array}
$$

PROOF:

We discard the $k = 1$ case, since it contains a conjunct $\mathit{in}[i][1][2] = a$ that contradicts the hypothesis $a \in \mathit{Addr} \setminus \{\mathit{Head}(\mathit{in}[i])[2]\}$.

$$
\begin{array}{l}
\langle 13\rangle 1.\ \land\ i \in \mathit{Proc} \land \mathit{HCacheUpdate}(i) \\
\qquad \land\ a \in \mathit{Addr} \setminus \{\mathit{Head}(\mathit{in}[i])[2]\} \\
\qquad \land\ \exists k \in 2 \,..\, \mathit{Len}(\mathit{in}[i]) \ : \\
\qquad\qquad \land\ \mathit{NotRefresh}_i(k) \\
\qquad\qquad \land\ \mathit{in}[i][k][1] = \mathit{mem}[a] \\
\qquad\qquad \land\ \mathit{in}[i][k][2] = a \\
\qquad\qquad \land\ \forall j \in k+1 \,..\, \mathit{Len}(\mathit{in}[i]) \ :\ \lor\ \mathit{in}[i][j][2] \neq a \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ \lor\ \mathit{in}[i][j][1] = \mathit{mem}[a] \\
\qquad \Rightarrow \exists k \in 1 \,..\, \mathit{Len}(\mathit{in}'[i]) \ : \\
\qquad\qquad \land\ \mathit{NotRefresh}_i{}'(k) \\
\qquad\qquad \land\ \mathit{in}'[i][k][1] = \mathit{mem}'[a] \\
\qquad\qquad \land\ \mathit{in}'[i][k][2] = a \\
\qquad\qquad \land\ \forall j \in k+1 \,..\, \mathit{Len}(\mathit{in}'[i]) \ :\ \lor\ \mathit{in}'[i][j][2] \neq a \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \lor\ \mathit{in}'[i][j][1] = \mathit{mem}'[a]
\end{array}
$$

PROOF:

We prove it without the k quantifiers, so we can use existential generalisation to obtain the quantifiers. We are careful to use the right value, namely $k-1$, in the consequent.

LET: k : CONSTANT

$$
\begin{array}{l}
\langle 14\rangle 1.\ \land\ i \in \mathit{Proc} \land \mathit{HCacheUpdate}(i) \\
\qquad \land\ a \in \mathit{Addr} \setminus \{\mathit{Head}(\mathit{in}[i])[2]\} \\
\qquad \land\ k \in 2 \,..\, \mathit{Len}(\mathit{in}[i]) \\
\qquad \land\ \mathit{NotRefresh}_i(k) \\
\qquad \land\ \mathit{in}[i][k][1] = \mathit{mem}[a] \\
\qquad \land\ \mathit{in}[i][k][2] = a \\
\qquad \land\ \forall j \in k+1 \,..\, \mathit{Len}(\mathit{in}[i]) \ :\ \lor\ \mathit{in}[i][j][2] \neq a \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ \lor\ \mathit{in}[i][j][1] = \mathit{mem}[a]
\end{array}
$$

$$\Rightarrow \land\ (k-1) \in 1\,..\,Len(in'[i])$$
$$\land\ NotRefresh_i{}'(k-1)$$
$$\land\ in'[i][k-1][1] = mem'[a]$$
$$\land\ in'[i][k-1][2] = a$$
$$\land\ \forall j \in k\,..\,Len(in'[i])\ :\ \lor\ in'[i][j][2] \neq a \quad \lor\ in'[i][j][1] = mem'[a]$$

PROOF:

We fiddle with the quantifier $\forall j \in k\,..\,Len(in'[i])$ in the consequent to get it into the right form.

⟨15⟩1. $\land\ i \in Proc \land HCacheUpdate(i)$
$\land\ a \in Addr \setminus \{Head(in[i])[2]\}$
$\land\ k \in 2\,..\,Len(in[i])$
$\land\ NotRefresh_i(k)$
$\land\ in[i][k][1] = mem[a]$
$\land\ in[i][k][2] = a$
$\land\ \forall j \in k+1\,..\,Len(in[i])\ :\ \lor\ in[i][j][2] \neq a \quad \lor\ in[i][j][1] = mem[a]$
$\Rightarrow \land\ (k-1) \in 1\,..\,Len(in'[i])$
$\land\ NotRefresh_i{}'(k-1)$
$\land\ in'[i][k-1][1] = mem'[a]$
$\land\ in'[i][k-1][2] = a$
$\land\ \forall\, l \in k+1\,..\,Len(in[i])\ :\ \lor\ in[i][l][2] \neq a \quad \lor\ in[i][l][1] = mem[a]$

PROOF: The first conjunct of the consequent follows by arithmetic and the definition of $s\,..\,t$, the second, third and fourth by ⟨7⟩5, along with $mem'[a] = mem[a]$. The final is simply α-equivalent to the last hypothesis. □

⟨15⟩2. Q.E.D.

PROOF: Follows immediately from ⟨15⟩1 by using ⟨7⟩5 to substitute $in'[i][l-1]$for $in[i][l]$, the definition of $HCacheUpdate$ to substitute $mem'[a]$ for $mem[a]$, recalculating the range of $l-1$, and finally substituting j for $l-1$. □

⟨14⟩2. Q.E.D.

PROOF: Follows from ⟨14⟩1 by existential generalisation over $k-1$ in the consequent, and k in the antecedent, and by predicate logic. □

⟨13⟩2. Q.E.D.

PROOF: Follows directly from ⟨13⟩1 by adding a disjunct that contradicts other hypotheses (by asserting $in[i][1][2] = a$) to the last conjunct of the antecedent. □

⟨12⟩2. Q.E.D.

PROOF: Follows directly from ⟨12⟩1 using ⟨6⟩2. □

⟨11⟩2. Q.E.D.

PROOF: Immediate from ⟨11⟩1 by substituting "*LastIsMem*" and "*LastIsMem*′" for their definiens. □

Second, the proof for $a = Head(in[i])[2]$

⟨10⟩2. $\land\ i \in Proc$
$\land\ HCacheUpdate(i)$
$\land\ LastIsMem(Head(in[i])[2], i)$

$$\begin{array}{l}\Rightarrow \lor\ LastIsMem'(Head(in[i])[2], i) \\ \quad\ \lor\ NoMemWrite(Head(in[i])[2], i)\end{array}$$

PROOF:

We expand the definition of *LastIsMem* and *NoMemWrite*.

$\langle 11\rangle 1.$ $$\begin{array}{l} \land\ i \in Proc \land HCacheUpdate(i) \\ \land\ \exists\, k \in 1 \,..\, Len(in[i]) : \\ \qquad \land\ NotRefresh_i(k) \\ \qquad \land\ in[i][k][1] = mem[Head(in[i])[2]]] \\ \qquad \land\ in[i][k][2] = Head(in[i])[2] \\ \qquad \land\ \forall\, j \in k+1 \,..\, Len(in[i]) : \lor\ in[i][j][2] \neq Head(in[i])[2] \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \lor\ in[i][j][1] = mem[Head(in[i])[2]] \\ \Rightarrow \\ \quad \lor \quad \land\ i \in Proc \\ \qquad\ \ \land\ vc'[i][Head(in[i])[2]] = mem'[Head(in[i])[2]] \\ \qquad\ \ \land\ \forall\, k \in 1 \,..\, Len(in'[i]) : \lor\ in'[i][k][2] \neq Head(in[i])[2] \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \lor\ in'[i][k][1] = mem'[Head(in[i])[2]] \\ \quad \lor\ \land\ i \in Proc \\ \qquad \land\ \exists\, k \in 1 \,..\, Len(in'[i]) : \\ \qquad\qquad \land\ NotRefresh_i{}'(k) \\ \qquad\qquad \land\ in'[i][k][1] = mem'[Head(in[i])[2]]] \\ \qquad\qquad \land\ in'[i][k][2] = Head(in[i])[2] \\ \qquad\qquad \land\ \forall\, j \in k+1 \,..\, Len(in'[i]) : \lor\ in'[i][j][2] \neq Head(in[i])[2] \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad \lor\ in'[i][j][1] = mem'[Head(in[i])[2]] \end{array}$$

PROOF:

We split the k quantifier into the $k = 1$ and $k = 2 \,..\, Len(in[i])$ cases.

$\langle 12\rangle 1.$ $$\begin{array}{l} \land\ i \in Proc \land HCacheUpdate(i) \\ \land\ \lor\ \land\ NotRefresh_i(1) \\ \qquad \land\ in[i][1][1] = mem[Head(in[i])[2]]] \\ \qquad \land\ in[i][1][2] = Head(in[i])[2] \\ \qquad \land\ \forall\, j \in 2 \,..\, Len(in[i]) : \lor\ in[i][j][2] \neq Head(in[i])[2] \\ \qquad\qquad\qquad\qquad\qquad\qquad \lor\ in[i][j][1] = mem[Head(in[i])[2]] \\ \quad \lor\ \exists\, k \in 2 \,..\, Len(in[i]) : \\ \qquad\qquad \land\ NotRefresh_i(k) \\ \qquad\qquad \land\ in[i][k][1] = mem[Head(in[i])[2]]] \\ \qquad\qquad \land\ in[i][k][2] = Head(in[i])[2] \\ \qquad\qquad \land\ \forall\, j \in k+1 \,..\, Len(in[i]) : \lor\ in[i][j][2] \neq Head(in[i])[2] \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad \lor\ in[i][j][1] = mem[Head(in[i])[2]] \\ \Rightarrow \\ \quad \lor\ \land\ i \in Proc \\ \qquad \land\ vc'[i][Head(in[i])[2]] = mem'[Head(in[i])[2]] \\ \qquad \land\ \forall\, k \in 1 \,..\, Len(in'[i]) : \lor\ in'[i][k][2] \neq Head(in[i])[2] \\ \qquad\qquad\qquad\qquad\qquad\qquad \lor\ in'[i][k][1] = mem'[Head(in[i])[2]] \\ \quad \lor\ \land\ i \in Proc \\ \qquad \land\ \exists\, k \in 1 \,..\, Len(in'[i]) : \\ \qquad\qquad \land\ NotRefresh_i{}'(k) \\ \qquad\qquad \land\ in'[i][k][1] = mem'[Head(in[i])[2]]] \\ \qquad\qquad \land\ in'[i][k][2] = Head(in[i])[2] \\ \qquad\qquad \land\ \forall\, j \in k+1 \,..\, Len(in'[i]) : \lor\ in'[i][j][2] \neq Head(in[i])[2] \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad \lor\ in'[i][j][1] = mem'[Head(in[i])[2]] \end{array}$$

PROOF:

We split the proof into two parts, according to the disjunction in the antecedent, to which cases the disjuncts in the consequent correspond. First, the case $k = 1$.

$$\begin{array}{l} \langle 13\rangle 1.\ \wedge\ i \in \mathit{Proc} \wedge \mathit{HCacheUpdate}(i) \\ \qquad \wedge\ \mathit{NotRefresh}_i(1) \\ \qquad \wedge\ \mathit{in}[i][1][1] = \mathit{mem}[\mathit{Head}(\mathit{in}[i])[2]] \\ \qquad \wedge\ \mathit{in}[i][1][2] = \mathit{Head}(\mathit{in}[i])[2] \\ \qquad \wedge\ \forall j \in 2\,..\,\mathit{Len}(\mathit{in}[i]) :\ \vee\ \mathit{in}[i][j][2] \neq \mathit{Head}(\mathit{in}[i])[2] \\ \qquad\qquad\qquad\qquad\qquad\quad\ \vee\ \mathit{in}[i][j][1] = \mathit{mem}[\mathit{Head}(\mathit{in}[i])[2]] \\ \qquad \Rightarrow \wedge\ i \in \mathit{Proc} \\ \qquad\quad\ \wedge\ \mathit{vc}'[i][\mathit{Head}(\mathit{in}[i])[2]] = \mathit{mem}'[\mathit{Head}(\mathit{in}[i])[2]] \\ \qquad\quad\ \wedge\ \forall k \in 1\,..\,\mathit{Len}(\mathit{in}'[i]) :\ \vee\ \mathit{in}'[i][k][2] \neq \mathit{Head}(\mathit{in}[i])[2] \\ \qquad\qquad\qquad\qquad\qquad\qquad\ \vee\ \mathit{in}'[i][k][1] = \mathit{mem}'[\mathit{Head}(\mathit{in}[i])[2]] \end{array}$$

PROOF: The first conjunct of the consequent follows from the first of the antecedent; the second from the third, using that *mem* is unchanged by *HCacheUpdate*; the third from *mem* being unchanged, and by $\langle 7\rangle 2$ with a variable change, as in $\langle 9\rangle 2$. $\square$

Now the case $k \in 2\,..\,\mathit{Len}(\mathit{in}[i])$.

$$\begin{array}{l} \langle 13\rangle 2.\ \wedge\ i \in \mathit{Proc} \wedge \mathit{HCacheUpdate}(i) \\ \qquad \wedge\ \exists k \in 2\,..\,\mathit{Len}(\mathit{in}[i]) : \\ \qquad\qquad \wedge\ \mathit{NotRefresh}_i(k) \\ \qquad\qquad \wedge\ \mathit{in}[i][k][1] = \mathit{mem}[\mathit{Head}(\mathit{in}[i])[2]] \\ \qquad\qquad \wedge\ \mathit{in}[i][k][2] = \mathit{Head}(\mathit{in}[i])[2] \\ \qquad\qquad \wedge\ \forall j \in k+1\,..\,\mathit{Len}(\mathit{in}[i]) :\ \vee\ \mathit{in}[i][j][2] \neq \mathit{Head}(\mathit{in}[i])[2] \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \vee\ \mathit{in}[i][j][1] = \mathit{mem}[\mathit{Head}(\mathit{in}[i])[2]] \\ \qquad \Rightarrow \wedge\ i \in \mathit{Proc} \\ \qquad\quad\ \wedge\ \exists k \in 1\,..\,\mathit{Len}(\mathit{in}'[i]) : \\ \qquad\qquad\quad \wedge\ \mathit{NotRefresh}_i{}'(k) \\ \qquad\qquad\quad \wedge\ \mathit{in}'[i][k][1] = \mathit{mem}'[\mathit{Head}(\mathit{in}[i])[2]] \\ \qquad\qquad\quad \wedge\ \mathit{in}'[i][k][2] = \mathit{Head}(\mathit{in}[i])[2] \\ \qquad\qquad\quad \wedge\ \forall j \in k+1\,..\,\mathit{Len}(\mathit{in}'[i]) :\ \vee\ \mathit{in}'[i][j][2] \neq \mathit{Head}(\mathit{in}[i])[2] \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ \vee\ \mathit{in}'[i][j][1] = \mathit{mem}'[\mathit{Head}(\mathit{in}[i])[2]] \end{array}$$

PROOF:

We instantiate the quantifiers in the antecedent with k and the consequent with $k-1$, as before.

LET: k : CONSTANT

$$\begin{array}{l} \langle 14\rangle 1.\ \wedge\ i \in \mathit{Proc} \wedge \mathit{HCacheUpdate}(i) \\ \qquad \wedge\ k \in 2\,..\,\mathit{Len}(\mathit{in}[i]) \\ \qquad \wedge\ \mathit{NotRefresh}_i(k) \\ \qquad \wedge\ \mathit{in}[i][k][1] = \mathit{mem}[\mathit{Head}(\mathit{in}[i])[2]] \\ \qquad \wedge\ \mathit{in}[i][k][2] = \mathit{Head}(\mathit{in}[i])[2] \\ \qquad \wedge\ \forall j \in k+1\,..\,\mathit{Len}(\mathit{in}[i]) :\ \vee\ \mathit{in}[i][j][2] \neq \mathit{Head}(\mathit{in}[i])[2] \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad\ \vee\ \mathit{in}[i][j][1] = \mathit{mem}[\mathit{Head}(\mathit{in}[i])[2]] \end{array}$$

$$\begin{array}{l} \Rightarrow \land\ i \in \mathit{Proc} \\ \quad \land\ (k-1) \in 1 \,..\, \mathit{Len}(\mathit{in}'[i]) \\ \quad \land\ \mathit{NotRefresh}_i{}'(k-1) \\ \quad \land\ \mathit{in}'[i][k-1][1] = \mathit{mem}'[\mathit{Head}(\mathit{in}[i])[2]] \\ \quad \land\ \mathit{in}'[i][k-1][2] = \mathit{Head}(\mathit{in}[i])[2] \\ \quad \land\ \forall j \in k \,..\, \mathit{Len}(\mathit{in}'[i]) \ :\ \lor\ \mathit{in}'[i][j][2] \neq \mathit{Head}(\mathit{in}[i])[2] \\ \qquad\qquad\qquad\qquad\qquad\quad\ \lor\ \mathit{in}'[i][j][1] = \mathit{mem}'[\mathit{Head}(\mathit{in}[i])[2]] \end{array}$$

PROOF:

The conjuncts in the consequent all follow from the corresponding ones in the antecedent using $\langle 7\rangle 4$ with arithmetic calculation on $2 \,..\, \mathit{Len}(\mathit{in}[i])$, and $\langle 7\rangle 5$. We handle the quantifiers in this substep.

$\langle 15\rangle 1.$
$$\begin{array}{l} \land\ i \in \mathit{Proc} \land \mathit{HCacheUpdate}(i) \\ \land\ k \in 2 \,..\, \mathit{Len}(\mathit{in}[i]) \\ \land\ \mathit{NotRefresh}_i(k) \\ \land\ \mathit{in}[i][k][1] = \mathit{mem}[\mathit{Head}(\mathit{in}[i])[2]] \\ \land\ \mathit{in}[i][k][2] = \mathit{Head}(\mathit{in}[i])[2] \\ \land\ j \in k+1 \,..\, \mathit{Len}(\mathit{in}[i]) \\ \land\ \lor\ \mathit{in}[i][j][2] \neq \mathit{Head}(\mathit{in}[i])[2] \\ \quad\ \lor\ \mathit{in}[i][j][1] = \mathit{mem}[\mathit{Head}(\mathit{in}[i])[2]] \\ \Rightarrow \land\ i \in \mathit{Proc} \\ \quad \land\ (k-1) \in 1 \,..\, \mathit{Len}(\mathit{in}'[i]) \\ \quad \land\ \mathit{NotRefresh}_i{}'(k-1) \\ \quad \land\ \mathit{in}'[i][k-1][1] = \mathit{mem}'[\mathit{Head}(\mathit{in}[i])[2]] \\ \quad \land\ \mathit{in}'[i][k-1][2] = \mathit{Head}(\mathit{in}[i])[2] \\ \quad \land\ (j-1) \in k \,..\, \mathit{Len}(\mathit{in}'[i]) \\ \quad \land\ \lor\ \mathit{in}'[i][j-1][2] \neq \mathit{Head}(\mathit{in}[i])[2] \\ \qquad\ \lor\ \mathit{in}'[i][j-1][1] = \mathit{mem}'[\mathit{Head}(\mathit{in}[i])[2]] \end{array}$$

PROOF: Conjuncts in the consequent all follow from the corresponding conjuncts in the antecedent using some arithmetic, and $\langle 7\rangle 4$ and $\langle 7\rangle 5$. □

$\langle 15\rangle 2.$ Q.E.D.

PROOF: Immediate from $\langle 15\rangle 1$ by universal generalisation over j, then substituting a new variable l for $j-1$ in the consequent, and then by α-equivalence. □

$\langle 14\rangle 2.$ Q.E.D.

PROOF: Immediate by existential generalisation over k in the antecedent and over a variable in the consequent that is substituted first for $k-1$, in $\langle 13\rangle 2$. □

$\langle 13\rangle 3.$ Q.E.D.

PROOF: Follows by propositional logic from $\langle 13\rangle 1$ and $\langle 13\rangle 1$. □

$\langle 12\rangle 2.$ Q.E.D.

PROOF: Follows from $\langle 12\rangle 1$ by $\langle 6\rangle 2$ on the hypothesis. □

$\langle 11\rangle 2.$ Q.E.D.

PROOF: $\langle 10\rangle 2$ is a rewrite of $\langle 11\rangle 1$. □

$\langle 10\rangle 3.$ Q.E.D.

PROOF: Immediate from $\langle 10\rangle 1$ and $\langle 10\rangle 2$ using $\langle 6\rangle 2$. □

Next, *FirstIsVC* is preserved. But it isn't always. Also, we need the hypotheses *BetweenMemWrites* and *LastIsMem*.

$\langle 9\rangle 4.$ $\land\ i \in \mathit{Proc} \land \mathit{HCacheUpdate}(i)$
$\land\ \mathit{BetweenMemWrites}(i)$
$\land\ \mathit{LastIsMem}(a, i)$
$\land\ \mathit{FirstIsVC}(a, i)$
$\Rightarrow \lor\ \mathit{FirstIsVC}'(a, i)$
$\lor\ \mathit{NoMemWrite}'(a, i))$

PROOF:

This is split into two steps: $a = \mathit{Head}(\mathit{in}[i])[2]$ and $a \in \mathit{Addr} \setminus \{\mathit{Head}(\mathit{in}[i])[2]\}$. First, $a \in \mathit{Addr} \setminus \{\mathit{Head}(\mathit{in}[i])[2]\}$

$\langle 10\rangle 1.$ $\land\ i \in \mathit{Proc} \land \mathit{HCacheUpdate}(i)$
$\land\ a \in \mathit{Addr} \setminus \{\mathit{Head}(\mathit{in}[i])[2]\}$
$\land\ \mathit{FirstIsVC}(a, i)$
$\Rightarrow \mathit{FirstIsVC}'(a, i)$

PROOF:

We unpack the definitions of $\mathit{FirstIsVC}(a, i)$ and $\mathit{FirstIsVC}'(a, i)$.

$$
\begin{aligned}
\langle 11\rangle 1.\ & \land\ i \in \mathit{Proc} \land \mathit{HCacheUpdate}(i)\\
& \land\ a \in \mathit{Addr} \setminus \{\mathit{Head}(\mathit{in}[i])[2]\}\\
& \land\ \exists\, k \in 1 \,..\, \mathit{Len}(\mathit{in}[i]) \ :\\
& \qquad \land\ \mathit{NotRefresh}_i(k)\\
& \qquad \land\ \mathit{in}[i][k][2] = a\\
& \qquad \land\ \forall\, j \in 1 \,..\, k-1 \ :\\
& \qquad\qquad \lor\ \mathit{in}[i][j][2] \neq a\\
& \qquad\qquad \lor\ \mathit{in}[i][j][1] = \mathit{vc}[i][a]\\
& \Rightarrow \exists\, k \in 1 \,..\, \mathit{Len}(\mathit{in}'[i]) \ :\\
& \qquad \land\ \mathit{NotRefresh}'_i(k)\\
& \qquad \land\ \mathit{in}'[i][k][2] = a\\
& \qquad \land\ \forall\, j \in 1 \,..\, k-1 \ :\ \lor\ \mathit{in}'[i][j][2] \neq a\\
& \qquad \lor\ \mathit{in}'[i][j][1] = \mathit{vc}'[i][a]
\end{aligned}
$$

PROOF:

We split the quantifier in the hypothesis into $k = 1$ and $k \in 2 \,..\, \mathit{Len}(\mathit{in}[i])$.

$$
\begin{aligned}
\langle 12\rangle 1.\ & \land\ i \in \mathit{Proc} \land \mathit{HCacheUpdate}(i)\\
& \land\ a \in \mathit{Addr} \setminus \{\mathit{Head}(\mathit{in}[i])[2]\}\\
& \land\ \lor\ \land\ \mathit{NotRefresh}_i(1)\\
& \qquad\quad \land\ \mathit{in}[i][1][2] = a\\
& \qquad\quad \land\ \forall\, j \in 1 \,..\, k-1 \ :\\
& \qquad\qquad\quad \lor\ \mathit{in}[i][j][2] \neq a\\
& \qquad\qquad\quad \lor\ \mathit{in}[i][j][1] = \mathit{vc}[i][a]\\
& \quad\ \lor\ \exists\, k \in 2 \,..\, \mathit{Len}(\mathit{in}[i]) \ :\\
& \qquad\quad \land\ \mathit{NotRefresh}_i(k)\\
& \qquad\quad \land\ \mathit{in}[i][k][2] = a\\
& \qquad\quad \land\ \forall\, j \in 1 \,..\, k-1 \ :\\
& \qquad\qquad\quad \lor\ \mathit{in}[i][j][2] \neq a\\
& \qquad\qquad\quad \lor\ \mathit{in}[i][j][1] = \mathit{vc}[i][a]
\end{aligned}
$$

$$\Rightarrow \exists\, k \in 1\,..\,Len(in'[i]) \; : \\ \quad \wedge\; NotRefresh'_i(k) \\ \quad \wedge\; in'[i][k][2] = a \\ \quad \wedge\; \forall\, j \in 1\,..\,k-1 \; : \; \vee\; in'[i][j][2] \neq a \\ \qquad\qquad\qquad\qquad\quad \vee\; in'[i][j][1] = vc'[i][a]$$

PROOF:

The hypothesis $in[i][1][2] = a$ for the case $k = 1$ is inconsistent with the hypothesis $HCacheUpdate(i) \wedge a \in Addr \setminus \{Head(in[i])[2]\}$ so we consider only the disjunct $k \in 2\,..\,Len(in[i])$.

$\langle 13\rangle 1.$ $\wedge\; i \in Proc \wedge HCacheUpdate(i)$
$$\wedge\; a \in Addr \setminus \{Head(in[i])[2]\} \\ \wedge\; \exists\, k \in 2\,..\,Len(in[i]) \; : \\ \quad \wedge\; NotRefresh_i(k) \\ \quad \wedge\; in[i][k][2] = a \\ \quad \wedge\; \forall\, j \in 1\,..\,k-1 \; : \\ \qquad \vee\; in[i][j][2] \neq a \\ \qquad \vee\; in[i][j][1] = vc[i][a] \\ \Rightarrow \exists\, k \in 1\,..\,Len(in'[i]) \; : \\ \quad \wedge\; NotRefresh'_i(k) \\ \quad \wedge\; in'[i][k][2] = a \\ \quad \wedge\; \forall\, j \in 1\,..\,k-1 \; : \; \vee\; in'[i][j][2] \neq a \\ \qquad\qquad\qquad\qquad\quad \vee\; in'[i][j][1] = vc'[i][a]$$

PROOF:

We instantiate the existential quantifier k in the antecedent and cunningly instantiate the quantifier k in the consequent with $k - 1$.

LET: k : CONSTANT

$\langle 14\rangle 1.$ $\wedge\; i \in Proc \wedge HCacheUpdate(i)$
$$\wedge\; a \in Addr \setminus \{Head(in[i])[2]\} \\ \wedge\; k \in 2\,..\,Len(in[i]) \; : \\ \wedge\; NotRefresh_i(k) \\ \wedge\; in[i][k][2] = a \\ \wedge\; \forall\, j \in 1\,..\,k-1 \; : \\ \quad \vee\; in[i][j][2] \neq a \\ \quad \vee\; in[i][j][1] = vc[i][a] \\ \Rightarrow \wedge\; (k-1) \in 1\,..\,Len(in'[i]) \\ \quad \wedge\; NotRefresh'_i(k-1) \\ \quad \wedge\; in'[i][k-1][2] = a \\ \quad \wedge\; \forall\, j \in 1\,..\,k-2 \; : \; \vee\; in'[i][j][2] \neq a \\ \qquad\qquad\qquad\qquad\quad \vee\; in'[i][j][1] = vc'[i][a]$$

PROOF:

We change the variable $l = (j + 1)$ in the consequent.

⟨15⟩1. $\wedge\ i \in Proc \wedge HCacheUpdate(i)$
$\wedge\ a \in Addr \setminus \{Head(in[i])[2]\}$
$\wedge\ k \in 2 \,..\, Len(in[i])\ :$
$\wedge\ NotRefresh_i(k)$
$\wedge\ in[i][k][2] = a$
$\wedge\ \forall\, j \in 1 \,..\, k-1\ :$
$\quad \vee\ in[i][j][2] \neq a$
$\quad \vee\ in[i][j][1] = vc[i][a]$
$\Rightarrow \wedge\ (k-1) \in 1 \,..\, Len(in'[i])$
$\quad \wedge\ NotRefresh'_i(k-1)$
$\quad \wedge\ in'[i][k-1][2] = a$
$\quad \wedge\ \forall\, l \in 2 \,..\, k-1\ :\ \vee\ in'[i][l-1][2] \neq a$
$\qquad\qquad\qquad\qquad \vee\ in'[i][l-1][1] = vc'[i][a]$

PROOF: Conjuncts in the consequent follow from corresponding conjuncts in the antecedent, using some arithmetic, and ⟨7⟩4 and ⟨7⟩5, and weakening the quantifier in the consequent. □

⟨15⟩2. Q.E.D.

PROOF: Follows directly from ⟨15⟩1 by the substitution $j = (l-1)$ in the quantifier in the consequent. □

⟨14⟩2. Q.E.D.

PROOF: Follows by existential generalisation over $k-1$ in the consequent of ⟨14⟩1, and universal generalisation over k in the antecedent. □

⟨13⟩2. Q.E.D.

PROOF: Follows by propositional logic from ⟨13⟩1 by adding a disjunct to the antecedent which is inconsistent with the antecedent. □

⟨12⟩2. Q.E.D.

PROOF: Follows directly from ⟨6⟩2 on the disjunct in the antecedent of ⟨12⟩1. □

⟨11⟩2. Q.E.D.

PROOF: Follows substituting the symbols "$FirstIsVC$" and "$FirstIsVC'$" for their definiens in ⟨11⟩1. □

Second, the proof for $a = Head(in[i])[2]$.
This requires all the hypotheses.

⟨10⟩2. $\wedge\ i \in Proc$
$\wedge\ HCacheUpdate(i)$
$\wedge\ BetweenMemWrites(Head(in[i])[2])$
$\wedge\ FirstIsVC(Head(in[i])[2], i)$
$\wedge\ LastIsMem(Head(in[i])[2], i)$
$\Rightarrow \vee\ FirstIsVC'(Head(in[i])[2], i)$
$\quad \vee\ NoMemWrite'(Head(in[i])[2], i))$

PROOF:

We case-split on $\forall\, h \in 2 \,..\, Len(in[i])\ :\ \neg NotRefresh_i(h)$ and $\exists\, h \in 2 \,..\, Len(in[i])\ :\ NotRefresh_i(h)$.
First, $\forall\, h \in 2 \,..\, Len(in[i])\ :\ \neg NotRefresh_i(h)$

⟨11⟩1. $\wedge\ i \in Proc$
$\wedge\ HCacheUpdate(i)$
$\wedge\ \forall\, h \in 2 \,..\, Len(in[i])\ :\ \neg NotRefresh_i(h)$
$\wedge\ LastIsMem(Head(in[i])[2], i)$
$\Rightarrow NoMemWrite(Head(in[i])[2], i)$

PROOF:

We unpack the definition of *LastIsMem*.

$\langle 12 \rangle 1.$ $\land\ i \in Proc$
$\land\ HCacheUpdate(i)$
$\land\ \forall\, h \in 2\,..\,Len(in[i])\ :\ \neg NotRefresh_i(h)$
$\land\ \exists\, k \in 1\,..\,Len(in[i])\ :$
$NotRefresh_i(k)$
$in[i][k][1] = mem[in[i][1][2]]$
$in[i][k][2] = in[i][1][2]$
$\forall\, j \in (k+1)\,..\,Len(in[i])\ :\ \lor\ in[i][j][2] \neq Head(in[i])[2]$
$\qquad \lor\ in[i][j][1] = mem[Head(in[i])[2]]$
$\Rightarrow\ NoMemWrite(Head(in[i])[2], i)$

PROOF:

We remove the quantifier from k, observe and instantiate $k = 1$ because all other values are $\neg NotRefresh$ by hypothesis, and drop all the hypotheses we don't need.

$\langle 13 \rangle 1.$ $\land\ i \in Proc$
$\land\ HCacheUpdate(i)$
$\land\ in[i][1][1] = mem[Head(in[i])[2]]$
$\land\ \forall\, j \in 2\,..\,Len(in[i])\ :\ \lor\ in[i][j][2] \neq Head(in[i])[2]$
$\qquad \lor\ in[i][j][1] = mem[Head(in[i])[2]]$
$\Rightarrow\ \land\ i \in Proc$
$\quad \land\ vc'[i][Head(in[i])[2]] = mem[Head(in[i])[2]]$
$\quad \land\ \forall\, j \in 1\,..\,Len(in'[i])\ :\ \lor\ in'[i][j][2] \neq Head(in[i])[2]$
$\qquad \lor\ in'[i][j][1] = mem'[Head(in[i])[2]]$

PROOF: Follows directly by substitution of $vc'[i][Head(in[i])[2]]$ for $in[i][1][1]$ using the definitions of *Head* and *HCacheUpdate*, and by substitution of the variable $j \leftarrow (j-1)$ and $\langle 7 \rangle 5$ within the quantifier $\forall\, j$ to obtain the equivalent quantified expression in the conclusion. □

$\langle 13 \rangle 2.$ Q.E.D.

PROOF: Direct from $\langle 12 \rangle 1$ by adding hypotheses, and by substituting "*NoMemwrite*" for its definiens. □

$\langle 12 \rangle 2.$ Q.E.D.

PROOF: By universal generalisation over k from $\langle 11 \rangle 1$. □

Second, $\exists\, h \in 2\,..\,Len(in[i])\ :\ NotRefresh_i(h)$.

$\langle 11 \rangle 2.$ $\land\ i \in Proc$
$\land\ HCacheUpdate(i)$
$\land\ \exists\, h \in 2\,..\,Len(in[i])\ :\ NotRefresh_i(h)$
$\land\ FirstIsVC(Head(in[i])[2], i)$
$\land\ BetweenMemWrites(i)$
$\Rightarrow\ FirstIsVC'(Head(in[i])[2], i)$

PROOF:

We use the CHOOSE function to select the least such h.

LET: $l \ \triangleq\ \text{CHOOSE}\ h\ :\ \land\ h \in 2\,..\,Len(in[i])$
$\qquad \land\ NotRefresh_i(h)$
$\qquad \land\ \forall\, m \in 2\,..\,Len(in[i])\ :\ NotRefresh_i(m) \Rightarrow h \leq m$

$\langle 12\rangle 1.$ $\exists\, h \in 2\,..\,Len(in[i])\ :\ NotRefresh_i(h)$
$\Rightarrow \land\ l \in 2\,..\,Len(in[i])$
$\land\ NotRefresh_i(l)$
$\land\ \forall\, m \in 2\,..\,Len(in[i])\ :\ NotRefresh_i(m) \Rightarrow l \leq m$

PROOF: Immediate from the definition of CHOOSE and the well-founded property of the *Naturals*, that there is a least *Natural* satisfying any non-empty predicate of *Naturals*. □

$\langle 12\rangle 2.$ $\land\ i \in Proc$
$\land\ HCacheUpdate(i)$
$\land\ l \in 2\,..\,Len(in[i])$
$\land\ NotRefresh_i(h)$
$\land\ \forall\, m \in 2\,..\,Len(in[i])\ :\ NotRefresh_i(m) \Rightarrow l \leq m$
$\land\ FirstIsVC(Head(in[i])[2], i)$
$\land\ BetweenMemWrites(i)$
$\Rightarrow FirstIsVC'(Head(in[i])[2], i)$

PROOF:

We case split on $NotRefresh_i(1)$.
First, $NotRefresh_i(1)$. We instantiate *BetweenMemWrites* as $MemWritesFromTo(1, l, i)$.

$\langle 13\rangle 1.$ $\land\ i \in Proc$
$\land\ HCacheUpdate(i)$
$\land\ l \in 2\,..\,Len(in[i])$
$\land\ NotRefresh_i(l)$
$\land\ \forall\, m \in 2\,..\,Len(in[i])\ :\ NotRefresh_i(m) \Rightarrow l \leq m$
$\land\ NotRefresh_i(1)$
$\land\ MemWritesFromTo(1, l, i)$
$\Rightarrow \land\ NotRefresh'_i(l-1)$
$\land\ \forall\, j \in 1\,..\,l-1 \lor\ in'[i][j][2] \neq Head(in[i])[2]$
$\lor\ in'[i][j][1] = vc'[i][Head(in[i])[2]]$

PROOF: Immediate from $\langle 7\rangle 5$ and the definition of *HCacheUpdate*, by expanding the definition of $MemWritesFromTo(1, l, i)$, using Modus Ponens, then substituting $vc'[i][Head(in[i])[2]]$ for $in[i][1][1]$ in the consequent, and renaming the quantified variable. □

The second part for $\neg NotRefresh_i(1)$ follows from *FirstIsVC*.

$\langle 13\rangle 2.$ $\land\ i \in Proc$
$\land\ HCacheUpdate(i)$
$\land\ l \in 2\,..\,Len(in[i])$
$\land\ NotRefresh_i(l)$
$\land\ \forall\, m \in 2\,..\,Len(in[i])\ :\ NotRefresh_i(m) \Rightarrow l \leq m$
$\land\ \neg NotRefresh_i(1)$
$\land\ FirstIsVC(Head(in[i])[2], i)$
$\Rightarrow \land\ NotRefresh'_i(l-1)$
$\land\ \forall\, j \in 1\,..\,l-1 \lor\ in'[i][j][2] \neq Head(in[i])[2]$
$\lor\ in'[i][j][1] = vc'[i][Head(in[i])[2]]$

PROOF: Immediate from $\langle 7\rangle 5$ by renaming the quantified variable. □

$\langle 13\rangle 3.$ Q.E.D.

PROOF: Immediate from $\langle 13\rangle 1$ and $\langle 13\rangle 2$ by propositional logic and then substituting "$FirstIsVC'$" for its definiens in the conclusion. □

$\langle 12\rangle 3.$ Q.E.D.

PROOF: Immediate from $\langle 12 \rangle 1$ and $\langle 12 \rangle 2$ by propositional logic. □

$\langle 11 \rangle 3$. Q.E.D.

PROOF: Immediate from $\langle 11 \rangle 1$ $\langle 11 \rangle 2$ by propositional logic, using the case-split which is a theorem of predicate logic. □

$\langle 10 \rangle 3$. Q.E.D.

PROOF: Immediate from $\langle 10 \rangle 1$ and $\langle 10 \rangle 2$ using $\langle 6 \rangle 2$. □

$\langle 9 \rangle 5$. Q.E.D.

PROOF: Follows by propositional logic from $\langle 9 \rangle 1$, $\langle 9 \rangle 2$, $\langle 9 \rangle 3$, and $\langle 9 \rangle 4$. □

Finally, the invariant $CCInv(p)$ holds trivially for any process $p \neq i$.

$\langle 8 \rangle 4$. $\wedge\ i \in Proc \wedge HCacheUpdate(i)$
$\wedge\ p \in Proc$
$\wedge\ p \neq i$
$\wedge\ CCInv(p)$
$\Rightarrow CCInv'(p)$

PROOF: It is immediate from the definition of $HCacheUpdate$ that $in[p], vc[p], hin[p]$ and $c[p]$ are UNCHANGED by $HCacheUpdate(i)$. □

Hence we conclude that $CCInv$ holds under $HCacheUpdate(i)$, having shown the three components of $CCInv(i)$, and $CCInv(p)$ for $p \neq i$.

$\langle 8 \rangle 5$. Q.E.D.

PROOF: Follows immediately by propositional logic from $\langle 8 \rangle 1$, $\langle 8 \rangle 2$, $\langle 8 \rangle 3$ and $\langle 8 \rangle 4$. □

$\langle 7 \rangle 7$. Q.E.D.

PROOF: Immediate from $\langle 3 \rangle 3$, $\langle 7 \rangle 6$ and universal quantification. □

The invariant is preserved by a *HMemRead*

$\langle 6\rangle 7.$ $i \in Proc \land CCInv \land HMemRead(i) \Rightarrow CCInv'$

PROOF:

$\langle 7\rangle 1.$ $\land\ i \in Proc$
$\quad\land\ HMemRead(i)$
$\quad\land\ p \in Proc$
$\quad\land\ p \neq i$
$\quad\land\ CCInv(p)$
$\qquad\Rightarrow\quad CCInv'(p)$

PROOF: By the definition of $HMemRead$, $in[p], hin[p], vc, mem$ are all UNCHANGED . □

$\langle 7\rangle 2.$ $i \in Proc \land CCInv(i) \land HMemRead(i) \Rightarrow CCInv'(i)$

PROOF:

$\langle 8\rangle 1.$ $i \in Proc \land HMemRead(i) \Rightarrow$

$$\begin{aligned}
&\land\ c' = c\\
&\land\ vc' = vc\\
&\land\ mem' = mem\\
&\land\ out' = out\\
&\land\ hout' = out\\
&\land\ \forall j \in Proc \setminus \{i\} \ :\ hin'[j] = hin[j]\\
&\land\ \forall j \in Proc \setminus \{i\} \ :\ in'[j] = in[j]\\
&\land\ Len(hin'[i]) = Len(hin[i]) + 1\\
&\land\ Len(in'[i]) = Len(in[i]) + 1\\
&\land\ \forall j \in 1\,..\,Len(in[i]) \ :\ in'[i][j] = in[i][j]\\
&\land\ in'[i][Len(in'[i])][1] = mem'[in'[i][Len(in'[i])][2]]\\
&\land\ \forall j \in 1\,..\,Len(in[i]) \ :\ NotRefresh_i{}'(j) \equiv NotRefresh_i(j)\\
&\land\ \neg NotRefresh_i{}'(Len(in'[i]))\\
&\land\ Len(Len(in'[i])) = 2\\
&\land\ hin'[Len(hin'[i])][2] \neq \text{“Wr”}
\end{aligned}$$

PROOF: Immediate from the definition of $HMemRead$. □

$\langle 8\rangle 2.$ $i \in Proc \land CCInv(i) \land HMemRead(i) \Rightarrow CCInv.1'(i)$

PROOF: Follows by logic from $\langle 8\rangle 1$. □

$\langle 8\rangle 3.$ $i \in Proc \land CCInv(i) \land HMemRead(i) \Rightarrow CCInv.2'(i)$

PROOF: Immediate from $\langle 8\rangle 1$. □

$\langle 8\rangle 4.$ $i \in Proc \land CCInv(i) \land HMemRead(i) \Rightarrow CCInv.3'(i)$

PROOF:

$\langle 9\rangle 1.$ $i \in Proc \land HMemRead(i) \Rightarrow$

$$\forall j \in 1\,..\,Len(in'[i]) \ :\ \phi(j) \equiv (\forall j \in 1\,..\,Len(in[i]) \ :\ \phi(j) \land \phi(Len(in'[i]))$$

PROOF: This is the quantifier-split property, as in the safety part of the proof, using $Len(in'[i]) = Len(in[i]) + 1$ from $\langle 8\rangle 1$. □

$\langle 9\rangle 2.$ $\land\ i \in Proc$
$\quad\land\ HMemRead(i)$
$\quad\land\ BetweenMemWrites(i)$
$\qquad\Rightarrow\quad BetweenMemWrites'(i)$

PROOF:

We unpack the definition of $BetweenMemWrites$.

$\langle 10\rangle 1.$ $\land\ i \in Proc$
$\quad\land\ HMemRead(i)$
$\quad\land\ MemWritesFromTo(m, n, i)$
$\qquad\Rightarrow MemWritesFromTo'(m, n, i)$

PROOF: Follows immediately by $\langle 8\rangle 1$ and the definition of $1\,..\,Len(in[i])$ □

$\langle 10\rangle 2.$ $\wedge\ i \in \mathit{Proc}$
$\wedge\ \mathit{HMemRead}(i)$
$\wedge\ m \in 1 \,..\, \mathit{Len}(\mathit{in}'[i])$
$\Rightarrow \mathit{MemWritesFromTo}'(m, \mathit{Len}(\mathit{in}'[i]), i)$

PROOF: Follows immediately since the antecedent of $\mathit{MemWritesFromTo}'$ is false by $\langle 8\rangle 1$. □

$\langle 10\rangle 3.$ Q.E.D.

PROOF: Follows from $\langle 10\rangle 1$ and $\langle 10\rangle 2$ using $\langle 9\rangle 1$ and quantifying over m, n. □

$\langle 9\rangle 3.$ $\wedge\ \mathit{HMemRead}(i)$
$\wedge\ a \in \mathit{Addr}$
$\wedge\ \mathit{LastIsMem}(a, i)$
$\Rightarrow \mathit{LastIsMem}'(a, i)$

PROOF: Immediate from $\langle 8\rangle 1$ using $\langle 9\rangle 1$. □

$\langle 9\rangle 4.$ $\wedge\ \mathit{HMemRead}(i)$
$\wedge\ a \in \mathit{Addr}$
$\wedge\ \mathit{FirstIsVC}(a, i)$
$\Rightarrow \mathit{FirstIsVC}'(a, i)$

PROOF: Immediate from $\langle 8\rangle 1$ using $\langle 9\rangle 1$. □

$\langle 9\rangle 5.$ $\wedge\ \mathit{HMemRead}(i)$
$\wedge\ a \in \mathit{Addr}$
$\wedge\ \mathit{NoMemWrite}(a, i)$
$\Rightarrow \mathit{NoMemWrite}'(a, i)$

PROOF: Immediate from $\langle 8\rangle 1$ using $\langle 9\rangle 1$. □

$\langle 9\rangle 6.$ Q.E.D.

PROOF: Immediate from $\langle 9\rangle 2$, $\langle 9\rangle 3$, $\langle 9\rangle 4$ and $\langle 9\rangle 5$ using propositional logic and quantification over a. □

$\langle 8\rangle 5.$ Q.E.D.

PROOF: Immediate from $\langle 8\rangle 2$, $\langle 8\rangle 3$, and $\langle 8\rangle 4$ by propositional logic. □

$\langle 7\rangle 3.$ Q.E.D.

PROOF: Immediate from $\langle 7\rangle 1$ and $\langle 7\rangle 2$. □

The invariant is preserved by a *HCacheInval*

$\langle 6 \rangle 8.$ $CCInv \land HCacheInval(i) \Rightarrow CCInv'$

PROOF:

$\langle 7 \rangle 1.$ $CCInv(i) \land HCacheInval(i) \Rightarrow CCInv'(i)$

PROOF:

$\langle 8 \rangle 1.$ $CCInv(i) \land HCacheInval(i) \Rightarrow CCInv.1'(i)$

PROOF: Immediate from the definition of *HCacheInval* since $in, hin, out, hout$ are UNCHANGED . □

$\langle 8 \rangle 2.$ $CCInv(i) \land HCacheInval(i) \Rightarrow CCInv.2'(i)$

PROOF:

$\langle 9 \rangle 1.$ $x \in Restrict(y) \land y \in Restrict(z) \Rightarrow x \in Restrict(z)$

PROOF: Property of *Restrict* assumed without further proof. □

$\langle 9 \rangle 2.$ Q.E.D.

Follows from the definition of *HCacheInval* and $\langle 9 \rangle 1$ with $x \leftarrow c'[i]$, $y \leftarrow c[i]$ and $z = vc'[i] = vc[i]$ from the definition of *HCacheInval*. □

$\langle 8 \rangle 3.$ $CCInv(i) \land HCacheInval(i) \Rightarrow CCInv.3'(i)$

PROOF: Immediate from the definition of *HCacheInval* since in, hin, mem, vc are UNCHANGED . □

$\langle 8 \rangle 4.$ Q.E.D.

PROOF: Immediate from $\langle 8 \rangle 1$,$\langle 8 \rangle 2$ and $\langle 8 \rangle 3$. □

$\langle 7 \rangle 2.$
$$\begin{array}{l} \land\ HCacheInval(i) \\ \land\ p \in Proc \\ \land\ p \neq i \\ \land\ CCInv(p) \\ \quad \Rightarrow CCInv'(p) \end{array}$$

PROOF: Immediate from the definition of *HCacheInval*, since $in, hin, mem, vc, c[p]$ are UNCHANGED . □

$\langle 7 \rangle 3.$ Q.E.D.

PROOF: Immediate from $\langle 7 \rangle 1$ and $\langle 7 \rangle 2$. □

That completes the proof of the invariant action by action.

$\langle 6 \rangle 9.$ Q.E.D.

PROOF: $\langle 5 \rangle 1$ follows by propositional logic and existential quantification over i from $\langle 6 \rangle 3$, $\langle 6 \rangle 4$, $\langle 6 \rangle 5$, $\langle 6 \rangle 6$, $\langle 6 \rangle 7$, and $\langle 6 \rangle 8$. □

So one may use TLA rule INV1 to infer the $\Box$ conclusion.

$\langle 5 \rangle 2.$ Q.E.D.

PROOF: Immediate from $\langle 5 \rangle 1$ by TLA rule INV1. □

And finally, trivial propositional logic to conclude the invariant holds.

$\langle 4 \rangle 3.$ Q.E.D.

PROOF: Follows by propositional logic from $\langle 4 \rangle 1$ and $\langle 4 \rangle 2$. □

We now use the invariant and the safety property of *LazyCache* (actually, of the version with history variables) to prove the safety property of *CCache*. First, a general logical property of sets.

$\langle 3\rangle 6.$ $\forall x \in S : \phi \land T \subset S \Rightarrow \forall x \in T : \phi$

PROOF: This is general logic and set theory, which we assume without further proof. □

$\langle 3\rangle 7.$ $\land \Box \mathit{CCInv}$
$\quad \land \Box[\exists i \in \mathit{Proc} : \mathit{HOp}(i)]_{\mathit{hvars}}$
$\quad \Rightarrow \Box[\overline{\exists i \in \mathit{Proc} : \mathit{CC}(cc, cin, cout).\mathit{Next}(i)}]_{\overline{\langle cc, cin, cout, ch\rangle}}$

PROOF:

$\langle 4\rangle 1.$ $\Box \mathit{CCInv} \Rightarrow \Box[\exists i \in \mathit{Proc} : \mathit{HOp}(i)]_{\mathit{hvars}} \equiv \Box[\exists i \in \mathit{Proc} : \mathit{HOp}(i) \land \mathit{CCInv} \land \mathit{CCInv}']_{\mathit{hvars}}$

PROOF: Immediate from TLA rule INV2. □

$\langle 4\rangle 2.$ $\Box[\land \exists i \in \mathit{Proc} : \mathit{HOp}(i)$
$\quad \land \mathit{CCInv}$
$\quad \land \mathit{CCInv}']_{\mathit{hvars}}$
$\quad \Rightarrow \Box[\overline{\exists i \in \mathit{Proc} : \mathit{CC}(cc, cin, cout).\mathit{Next}(i)}]_{\overline{\langle cc, cin, cout, ch\rangle}}$

PROOF:

$\langle 5\rangle 1.$ $[\land \exists i \in \mathit{Proc} : \mathit{HOp}(i)$
$\quad \land \mathit{CCInv}$
$\quad \land \mathit{CCInv}']_{\mathit{hvars}}$
$\quad \Rightarrow [\overline{\exists i \in \mathit{Proc} : \mathit{CC}(cc, cin, cout).\mathit{Next}(i)}]_{\overline{\langle cc, cin, cout, ch\rangle}}$

PROOF:

$\langle 6\rangle 1.$ $\mathit{hvars}' = \mathit{hvars} \Rightarrow \overline{\langle cc, cin, cout, ch\rangle}' = \overline{\langle cc, cin, cout, ch\rangle}$

PROOF: Immediate since the latter are state functions of the former variables. □

HRead simulates $\overline{\mathit{CC}(cc, cin, cout).\mathit{Read}}$.

$\langle 6\rangle 2.$ $\land i \in \mathit{Proc}$
$\quad \land \mathit{HRead}(i, d, a)$
$\quad \land \mathit{CCInv}$
$\quad \land \mathit{CCInv}'$
$\quad \Rightarrow \overline{\mathit{CC}(cc, cin, cout).\mathit{Read}}(i, d, a)$

PROOF:

We expand the definitions.

$\langle 7\rangle 1.$ $\land i \in \mathit{Proc}$
$\quad \land \mathit{CCInv}$
$\quad \land \mathit{CCInv}'$
$\quad \land \mathit{out}[i] = \langle\,\rangle$
$\quad \land \forall j \in 1\,..\,\mathit{Len}(\mathit{in}[i]) : \mathit{Len}(\mathit{in}[i][j]) = 2$
$\quad \land c[i][a] = d$
$\quad \land \mathit{ChanOp}(ch[i], \langle i, \text{“}\mathsf{Rd}\text{”}, d, a\rangle)$
$\quad \land \forall j \in \mathit{Proc} \setminus \{i\} : \text{UNCHANGED } ch[j]$
$\quad \land \text{UNCHANGED } \langle c, \mathit{in}, \mathit{out}, \mathit{mem}, \mathit{hin}, \mathit{hout}, \mathit{vc}\rangle$
$\quad \Rightarrow \land \mathit{hout}[i] = \langle\,\rangle$
$\quad\quad \land \forall j \in (1\,..\,\mathit{Len}(\overline{cin}[i])) : \overline{cin}[i][j][1] \neq i$
$\quad\quad \land \mathit{vc}[i][a] = d$
$\quad\quad \land \mathit{ChanOp}(ch[i], \langle i, \text{“}\mathsf{Rd}\text{”}, d, a\rangle)$
$\quad\quad \land \forall j \in \mathit{Proc} \setminus \{i\} : \text{UNCHANGED } ch[j]$
$\quad\quad \land \text{UNCHANGED } \langle \mathit{vc}, \overline{cin}, \mathit{hout}\rangle$

PROOF: It follows from $\mathit{out}[i] = \langle\,\rangle$ that $\mathit{Len}(\mathit{out}[i]) = 0$ and by $\mathit{CCInv}.1$ that thus $\mathit{Len}(\mathit{hout}[i]) = 0$ and so that $\mathit{hout}[i] = \langle\,\rangle$. From $\forall j \in 1\,..\,\mathit{Len}(\mathit{in}[i]) : \mathit{Len}(\mathit{in}[i][j]) = 2$

it follows that $\forall j \in 1 \,..\, Len(hin[i]) : hin[i][j][1] = i \Rightarrow hin[i][j][2] \neq$ "Wr" and thus from $\langle 3 \rangle 6$ and the definition of SelectSeq and the definition of $\overline{cin}$ that $\forall j \in 1 \,..\, Len(\overline{cin}[i]) : \overline{cin}[i][j][1] \neq i$. The other conjuncts of the consequent all follow directly from hypotheses. □

$\langle 7 \rangle 2$. Q.E.D.

PROOF: Immediate from $\langle 7 \rangle 1$. □

HWrite simulates $\overline{CC(cc, cin, cout).Write}$.

$\langle 6 \rangle 3$. $i \in Proc \wedge HWrite(i) \Rightarrow \overline{CC(cc, cin, cout).Write}(i)$

PROOF:

$\langle 7 \rangle 1$.
$$\begin{array}{l}
\wedge\ i \in Proc \\
\wedge\ ChanOp(ch[i], \langle i, \text{"Wr"}, d, a \rangle) \\
\wedge\ out' = [out \text{ EXCEPT } ![i] = out[i] \circ \langle \langle d, a \rangle \rangle] \\
\wedge\ \forall j \in Proc \setminus \{i\} : \text{UNCHANGED } ch[j] \\
\wedge\ \text{UNCHANGED } \langle c, in, mem \rangle \\
\wedge\ hout' = [hout \text{ EXCEPT } ![i] = hout[i] \circ \langle \langle i, \text{"Wr"}, d, a \rangle \rangle] \\
\wedge\ \text{UNCHANGED } \langle vc, hin \rangle \\
\Rightarrow \wedge\ ChanOp(ch[i], \langle i, \text{"Wr"}, d, a \rangle) \\
\quad\ \wedge\ hout' = [hout \text{ EXCEPT } ![i] = hout[i] \circ \langle \langle i, \text{"Wr"}, d, a \rangle \rangle] \\
\quad\ \wedge\ \forall j \in Proc \setminus \{i\} : \text{UNCHANGED } ch[j] \\
\quad\ \wedge\ \text{UNCHANGED } \langle vc, \overline{cin} \rangle
\end{array}$$

PROOF: Immediate, observing that since $\overline{cin}$ is a state function of *hin*, UNCHANGED $hin \Rightarrow$ UNCHANGED $\overline{cin}$. □

$\langle 7 \rangle 2$. Q.E.D.

PROOF: Immediate from $\langle 7 \rangle 1$. □

HMemWrite simulates $\overline{CC(cc, cin, cout).MemWrite}$.

$\langle 6 \rangle 4$.
$$\begin{array}{l}
\wedge\ i \in Proc \\
\wedge\ HMemWrite(i) \\
\wedge\ CCInv \\
\wedge\ CCInv' \\
\Rightarrow \overline{CC(cc, cin, cout).MemWrite}(i)
\end{array}$$

PROOF:

$\langle 7 \rangle 1$.
$$\begin{array}{l}
\wedge\ CCInv.1(i) \\
\wedge\ CCInv.1'(i) \\
\wedge\ out[i] \neq \langle\,\rangle \\
\wedge\ out' = [out \text{ EXCEPT } ![i] = Tail(out[i])] \\
\wedge\ in' = [j \in Proc \mapsto in[j] \circ \textbf{if } j = i \textbf{ then } \langle Head(out[i]) \circ \langle \text{"*"} \rangle \rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{else } \langle Head(out[i]) \rangle \qquad] \\
\wedge\ mem' = [mem \text{ EXCEPT } ![Head(out[i])[2]] = Head(out[i])[1]] \\
\wedge\ \text{UNCHANGED } \langle c, ch \rangle \\
\wedge\ hin' = [j \in Proc \mapsto hin[j] \circ \langle \langle i, \text{"Wr"} \rangle \circ Head(out[i]) \rangle] \\
\wedge\ hout' = [hout \text{ EXCEPT } ![i] = Tail(hout[i])] \\
\wedge\ \text{UNCHANGED } vc \\
\Rightarrow \wedge\ hout[i] \neq \langle\,\rangle \\
\quad\ \wedge\ hout' = [hout \text{ EXCEPT } ![i] = Tail(hout[i])] \\
\quad\ \wedge\ \overline{cin}' = [j \in Proc \mapsto \overline{cin}[j] \circ \langle Head(hout[i]) \rangle] \\
\quad\ \wedge\ \text{UNCHANGED } \langle vc, ch \rangle
\end{array}$$

PROOF: From the hypotheses, $in'[i] = in[i] \circ \langle Head(out[i]) \circ \langle \text{"*"} \rangle \rangle$ thus $Len(Len(in'[i])) = 3$ thus by $CCInv.1'$, $hin'[i][Len(hin'[i])][2] =$ "Wr" thus by definition of *SelectSeq*, $\overline{cin}' = \overline{cin} \circ \langle Head(hout[i]) \rangle$. The other clauses follow directly from the hypotheses. □

$\langle 7\rangle 2$. Q.E.D.

PROOF: Immediate from $\langle 7\rangle 1$. □

HCacheUpdate simulates either $\overline{CC(cc, cin, cout).CacheUpdate}$, in case that $Head(in[i])$ is not a cache refresh, or a no-op in case it is.

$$\begin{array}{l}\langle 6\rangle 5.\ \land\ i \in Proc \\ \quad \land\ HCacheUpdate(i) \\ \quad \land\ CCInv \\ \quad \land\ CCInv' \\ \quad \Rightarrow [\overline{CC(cc, cin, cout).CacheUpdate}(i)]_{\overline{\langle cc, cin, cout, ch\rangle}}\end{array}$$

PROOF:

$$\begin{array}{l}\langle 7\rangle 1.\ \land\ i \in Proc \\ \quad \land\ CCInv \\ \quad \land\ CCInv' \\ \quad \land\ in[i] \neq \langle\,\rangle \\ \quad \land\ in' = [in \text{ EXCEPT } ![i] = Tail(in[i])] \\ \quad \land\ c' = [c \text{ EXCEPT } ![i][Head(in[i])[2]] = Head(in[i])[1]\,] \\ \quad \land\ \text{UNCHANGED } \langle out, ch, mem\rangle \\ \quad \land\ hin' = [hin \text{ EXCEPT } ![i] = Tail(hin[i])] \\ \quad \land\ vc' = [vc \text{ EXCEPT } ![i][Head(in[i])[2]] = Head(in[i])[1]] \\ \quad \land\ \text{UNCHANGED } hout \\ \quad \Rightarrow \land \lor \land\ \overline{cin}[i] \neq \langle\,\rangle \\ \qquad\qquad \land\ \overline{cin}' = [\overline{cin} \text{ EXCEPT } ![i] = Tail(\overline{cin}[i])] \\ \qquad\qquad \land\ vc' = [vc \text{ EXCEPT } ![i][Head(\overline{cin}[i])[4]] = Head(\overline{cin}[i])[3]] \\ \qquad\quad \lor\ \text{UNCHANGED } \langle \overline{cin}, vc\rangle \\ \qquad \land\ \text{UNCHANGED } \langle hout, ch\rangle\end{array}$$

PROOF:

We case-split on $NotRefresh(Head(in[i]))$. First, the case in which $NotRefresh(Head(in[i]))$.

$$\begin{array}{l}\langle 8\rangle 1.\ \land\ i \in Proc \\ \quad \land\ CCInv \\ \quad \land\ CCInv' \\ \quad \land\ in[i] \neq \langle\,\rangle \\ \quad \land\ NotRefresh(Head(in[i])) \\ \quad \land\ in' = [in \text{ EXCEPT } ![i] = Tail(in[i])] \\ \quad \land\ c' = [c \text{ EXCEPT } ![i][Head(in[i])[2]] = Head(in[i])[1]\,] \\ \quad \land\ \text{UNCHANGED } \langle out, ch, mem\rangle \\ \quad \land\ hin' = [hin \text{ EXCEPT } ![i] = Tail(hin[i])] \\ \quad \land\ vc' = [vc \text{ EXCEPT } ![i][Head(in[i])[2]] = Head(in[i])[1]] \\ \quad \land\ \text{UNCHANGED } hout \\ \quad \Rightarrow \land\ \overline{cin}[i] \neq \langle\,\rangle \\ \qquad \land\ \overline{cin}' = [\overline{cin} \text{ EXCEPT } ![i] = Tail(\overline{cin}[i])] \\ \qquad \land\ vc' = [vc \text{ EXCEPT } ![i][Head(\overline{cin}[i])[4]] = Head(\overline{cin}[i])[3]] \\ \qquad \land\ \text{UNCHANGED } \langle hout, ch\rangle\end{array}$$

PROOF: Immediate from the definition of *SelectSeq* that $\overline{cin} = Head(in[i]) \circ SelectSeq(Tail(hin[i]), P[2]$ "Rd") and the consequent follows by equality manipulation. □

Second, the case in which $\neg NotRefresh(Head(in[i]))$.

$\langle 8 \rangle 2.$ $\wedge\ i \in Proc$
$\wedge\ CCInv$
$\wedge\ CCInv'$
$\wedge\ in[i] \neq \langle\,\rangle$
$\wedge\ \neg NotRefresh(Head(in[i]))$
$\wedge\ in' = [in \text{ EXCEPT } ![i] = Tail(in[i])]$
$\wedge\ c' = [c \text{ EXCEPT } ![i][Head(in[i])[2]] = Head(in[i])[1]]$
$\wedge$ UNCHANGED $\langle out, ch, mem \rangle$
$\wedge\ hin' = [hin \text{ EXCEPT } ![i] = Tail(hin[i])]$
$\wedge\ vc' = [vc \text{ EXCEPT } ![i][Head(in[i])[2]] = Head(in[i])[1]]$
$\wedge$ UNCHANGED $hout$
$\Rightarrow \wedge$ UNCHANGED $\langle \overline{cin}, vc \rangle$
$\wedge$ UNCHANGED $\langle hout, ch \rangle$

PROOF: By *FirstIsVC*, $vc[Head(in[i])[2]] = Head(in[i])[1] = vc'[Head(in[i])[2]]$ by definition of *HCacheUpdate*. By *NoMemWrite*, the same follows with a little equality reasoning. By case-splitting on these two cases of the invariant, it follows that UNCHANGED vc. It follows from definition of *SelectSeq* and $\neg NotRefresh(Head(in[i]))$ that $SelectSeq(hin[i], P[2] \neq \text{"Rd"}) = SelectSeq(Tail(hin[i]), P[2] \neq \text{"Rd"})$ and thus UNCHANGED $\overline{cin}$. □

$\langle 8 \rangle 3.$ Q.E.D.
PROOF: Follows by propositional logic from $\langle 8 \rangle 1$ and $\langle 8 \rangle 2$. □

$\langle 7 \rangle 2.$ Q.E.D.
PROOF: Immediate from $\langle 7 \rangle 1$. □

HMemRead simulates a no-op.

$\langle 6 \rangle 6.$ $\wedge\ i \in Proc$
$\wedge\ HMemRead(i)$
$\wedge\ CCInv$
$\wedge\ CCInv'$
$\Rightarrow$ UNCHANGED $\overline{\langle cc, cin, cout, ch \rangle}$

PROOF:

$\langle 7 \rangle 1.$ $\wedge\ i \in Proc$
$\wedge\ \exists\, a \in Addr :$
$\quad in' = [in \text{ EXCEPT } ![i] = in[i] \circ \langle \langle mem[a], a \rangle \rangle]$
$\wedge$ UNCHANGED $\langle out, c, mem, ch \rangle$
$\wedge\ hin' = [hin \text{ EXCEPT } ![i] = hin[i] \circ \langle \langle i, \text{"Rd"} \rangle \rangle]$
$\wedge$ UNCHANGED $vc, hout$
$\Rightarrow$ UNCHANGED $\langle ch, vc, hout, \overline{cin} \rangle$

PROOF: Since $hin'[i][Len(hin'[i])][2] = \text{"Rd"}$, by definition of $\overline{cin}$, $hin'[i][Len(hin'[i])]$ is not appended to $\overline{cin}$ by $HMemRead(i)$ and thus UNCHANGED $\overline{cin}$. The other UNCHANGED variables are so stated in the antecedent. □

$\langle 7 \rangle 2.$ Q.E.D.
PROOF: Immediate from $\langle 7 \rangle 1$. □

HCacheInval simulates a no-op.

$\langle 6 \rangle 7.$ $\wedge\ i \in Proc$
$\wedge\ HCacheInval(i)$
$\wedge\ CCInv$
$\wedge\ CCInv'$
$\Rightarrow$ UNCHANGED $\overline{\langle cc, cin, cout, ch \rangle}$

PROOF:

$\langle 7 \rangle 1.$ $\wedge\ i \in \mathit{Proc}$
$\wedge\ c' \in \{[c \text{ EXCEPT } ![i] \in \mathit{Restrict}(c[i])]\}$
$\wedge$ UNCHANGED $\langle \mathit{in}, \mathit{out}, \mathit{mem}, \mathit{ch} \rangle$
$\wedge$ UNCHANGED $\langle \mathit{vc}, \mathit{hin}, \mathit{hout} \rangle$
$\Rightarrow$ UNCHANGED $\langle \mathit{ch}, \mathit{vc}, \mathit{hout}, \overline{\mathit{cin}} \rangle$

PROOF: Immediate. The variables *ch*, *vc*, *hout* are unchanged by hypothesis, and $\overline{\mathit{cin}}$ is a state function of *hin*, which is unchanged by the hypothesis. □

$\langle 7 \rangle 2.$ Q.E.D.

PROOF: Immediate from $\langle 7 \rangle 1$. □

From the individual step simulations, we conclude that the machine step-simulation is valid.

$\langle 6 \rangle 8.$ Q.E.D.

PROOF: Immediate from $\langle 6 \rangle 1$, $\langle 6 \rangle 2$, $\langle 6 \rangle 3$, $\langle 6 \rangle 4$, $\langle 6 \rangle 5$, $\langle 6 \rangle 6$, and $\langle 6 \rangle 7$ by propositional logic. □

This has shown the core mathematical part of the safety proof. Now we may apply TLA rules and propositional logic to show the safety assertion $\langle 3 \rangle 7$.

$\langle 5 \rangle 2.$ Q.E.D.

PROOF: Immediate by TLA rule STL4. □

$\langle 4 \rangle 3.$ Q.E.D.

PROOF: Follows by propositional logic from $\langle 4 \rangle 1$ and $\langle 4 \rangle 2$. □

We show the liveness properties by use of the TLA rule WF2.

$\langle 3\rangle 8.$ $\land\ \mathit{HInit} \land \Box[\exists\, i \in \mathit{Proc} : \mathit{HOp}(i)]_{\mathit{hvars}}$
$\quad \land\ \forall\, i \in \mathit{Proc} : \mathrm{WF}_{\mathit{hvars}}(\mathit{HCacheUpdate}(i))$
$\quad \land\ \forall\, i \in \mathit{Proc} : \mathrm{WF}_{\mathit{hvars}}(\mathit{HMemWrite}(i))$
$\quad \Rightarrow \land\ \forall\, i \in \mathit{Proc} : \overline{\mathrm{WF}_{\langle cc, cin, cout, ch\rangle}(\mathit{CC}(cc, cin, cout).\mathit{CacheUpdate}(i))}$
$\qquad \land\ \forall\, i \in \mathit{Proc} : \overline{\mathrm{WF}_{\langle cc, cin, cout, ch\rangle}(\mathit{CC}(cc, cin, cout).\mathit{MemWrite}(i))}$

PROOF:

Liveness of $\overline{\mathit{CC}(cc, cin, cout).\mathit{MemWrite}}$ is easy, since it is direcly simulated by $\mathit{HMemWrite}$, and since the *Enabled* conditions of both are the same. First, a general trivial lemma.

$\langle 4\rangle 1.$ $\mathit{CCInv} \land \mathit{CCInv}' \land \neg \text{UNCHANGED } \mathit{hvars} \Rightarrow \neg \text{UNCHANGED } \langle cc, cin, cout, ch\rangle$

PROOF: This is immediate from the definitions of the variables, and also that $hin, hout, vc$ change when in, out, c change, from $\mathit{CCInv} \land \mathit{CCInv}'$. □

$\langle 4\rangle 2.$ $\land\ \mathit{HInit} \land \Box[\exists\, i \in \mathit{Proc} : \mathit{HOp}(i)]_{\mathit{hvars}}$
$\quad \land\ \forall\, i \in \mathit{Proc} : \mathrm{WF}_{\mathit{hvars}}(\mathit{HMemWrite}(i))$
$\quad \Rightarrow$
$\quad \forall\, i \in \mathit{Proc} : \overline{\mathrm{WF}_{\langle cc, cin, cout, ch\rangle}(\mathit{CC}(cc, cin, cout).\mathit{MemWrite}(i))}$

PROOF:

First, we instantiate the quantifier

$\langle 5\rangle 1.$ $\land\ \mathit{HInit} \land \Box[\exists\, i \in \mathit{Proc} : \mathit{HOp}(i)]_{\mathit{hvars}}$
$\quad \land\ \mathrm{WF}_{\mathit{hvars}}(\mathit{HMemWrite}(p))$
$\quad \land\ p \in \mathit{Proc}$
$\quad \Rightarrow \land\ p \in \mathit{Proc}$
$\qquad \land\ \overline{\mathrm{WF}_{\langle cc, cin, cout, ch\rangle}(\mathit{CC}(cc, cin, cout).\mathit{MemWrite}(p))}$

PROOF:

$\langle 6\rangle 1.$ $\land\ p \in \mathit{Proc}$
$\quad \land\ \mathit{CCInv}$
$\quad \land\ \mathit{CCInv}'$
$\quad \land\ \langle \mathit{HMemWrite}(p)\rangle_{\mathit{hvars}}$
$\quad \Rightarrow \langle \overline{\mathit{CC}(cc, cin, cout).\mathit{MemWrite}(p)}\rangle_{\overline{\langle cc, cin, cout, ch\rangle}}$

PROOF: The substance of this was shown already in the step simulation in $\langle 3\rangle 7$, and the variable change follows from $\langle 4\rangle 1$. □

$\langle 6\rangle 2.$ $\land\ p \in \mathit{Proc}$
$\quad \land\ \mathit{CCInv}$
$\quad \land\ \mathit{CCInv}'$
$\quad \land\ \langle \mathit{HMemWrite}(p)\rangle_{\mathit{hvars}}$
$\quad \land\ \overline{\mathit{Enabled}\langle \mathit{CC}(cc, cin, cout).\mathit{MemWrite}(p)\rangle_{\langle cc, cin, cout, ch\rangle}}$
$\quad \Rightarrow \land\ \langle \mathit{HMemWrite}(p)\rangle_{\mathit{hvars}}$
$\qquad \land\ p \in \mathit{Proc}$
$\qquad \land\ \mathit{CCinv}$
$\qquad \land\ \mathit{CCInv}'$

PROOF: Immediate by propositional logic. □

$\langle 6\rangle 3.$ $\land\ \mathit{CCInv}$
$\quad \land\ \overline{\mathit{Enabled}\langle \mathit{CC}(cc, cin, cout).\mathit{MemWrite}(p)\rangle_{\langle cc, cin, cout, ch\rangle}}$
$\quad \Rightarrow \mathit{Enabled}\langle \mathit{HMemWrite}(p)\rangle_{\mathit{hvars}}$

PROOF:

$\langle 7\rangle 1.$ $\wedge\ CCInv$
$\wedge\ \exists\, x, y\ :$
$\quad \wedge\ hout[p] \neq \langle\,\rangle$
$\quad \wedge\ x = [hout\ \text{EXCEPT}\ ![p] = Tail(hout[p])]$
$\quad \wedge\ y = [j \in Proc \mapsto \overline{cin}[j] \circ \langle Head(hout[p])\rangle]$
$\quad \wedge\ \text{UNCHANGED}\ \langle vc, \overline{ch}\rangle$
$\wedge\ \neg\text{UNCHANGED}\ \langle vc, \overline{cin}, hout, ch\rangle$
$\Rightarrow \wedge\ \exists\, x, v, z, w, u\ :$
$\quad \wedge\ out[p] \neq \langle\,\rangle$
$\quad \wedge\ v = [out\ \text{EXCEPT}\ ![p] = Tail(out[p])]$
$\quad \wedge\ z = [j \in Proc\ \mapsto\ in[j] \circ \mathbf{if}\ j = p\ \mathbf{then}\ \langle Head(out[p]) \circ \langle \text{“}*\text{”}\rangle\rangle\ \mathbf{else}\ \langle Head(out[p])\rangle\ ]$
$\quad \wedge\ w = [mem\ \text{EXCEPT}\ ![Head(out[p])[2]] = Head(out[p])[1]]$
$\quad \wedge\ \text{UNCHANGED}\ \langle c, ch\rangle$
$\quad \wedge\ u = [j \in Proc \mapsto hin[j] \circ \langle\langle p, \text{“Wr”}\rangle\rangle]$
$\quad \wedge\ x = [hout\ \text{EXCEPT}\ ![p] = Tail(hout[p])]$
$\quad \wedge\ \text{UNCHANGED}\ vc$
$\wedge\ hvars' \neq hvars$

PROOF: All the values of the new variables are functions of the values of state variables, so they exist. The $\neg$UNCHANGED clause follows from the similar one in the hypothesis, and $out[i] \neq \langle\,\rangle$ follows from $hout[i] \neq \langle\,\rangle$. □

$\langle 7\rangle 2.$ Q.E.D.

PROOF: Immediate from $\langle 7\rangle 1$. □

$\langle 6\rangle 4.$ $\Box CCInv \Rightarrow \Diamond\Box CCInv$

PROOF: Immediate by the contraposition of STL2. □

$\langle 6\rangle 5.$ Q.E.D.

PROOF: For fixed p, immediate by TLA rule WF2 from $\langle 6\rangle 1$, $\langle 6\rangle 2$, and $\langle 6\rangle 3$ with

$\mathcal{A} \triangleq HMemWrite(p)$,
$f \triangleq hvars$,
$\mathcal{B} \triangleq p \in Proc \wedge \mathcal{A} \wedge CCInv \wedge CCInv'$,
$\mathcal{M} \triangleq \overline{CC(cc, cin, cout, ch).MemWrite(p)}$,
$g \triangleq \langle cc, cin, cout, ch\rangle$,
$P \triangleq p \in Proc \wedge CCInv$. □

$\langle 5\rangle 2.$ Q.E.D.

PROOF: Immediate by quantification over p from $\langle 5\rangle 1$. □

Liveness of $\overline{CC(cc, cin, cout).CacheUpdate}$ is a little more involved, since its *Enabled* condition and that of *HCacheUpdate* differ.

$\langle 4\rangle 3.$ $\wedge\ HInit \wedge \Box[\exists\, i \in Proc\ :\ HOp(i)]_{hvars}$
$\wedge\ \forall\, i \in Proc\ :\ \text{WF}_{hvars}(HCacheUpdate(i))$
$\Rightarrow \forall\, i \in Proc\ :\ \overline{\text{WF}_{\langle cc, cin, cout, ch\rangle}(CC(cc, cin, cout).CacheUpdate(i))}$

PROOF:

First, we instantiate the quantifier.

$\langle 5\rangle 1.$ $\wedge\ HInit \wedge \Box[\exists\, i \in Proc\ :\ HOp(i)]_{hvars}$
$\wedge\ \text{WF}_{hvars}(HCacheUpdate(p))$
$\wedge\ p \in Proc$
$\Rightarrow \wedge\ p \in Proc$
$\quad \wedge\ \overline{\text{WF}_{\langle cc, cin, cout, ch\rangle}(CC(cc, cin, cout).CacheUpdate(p))}$

PROOF:

First, a general formula that follows by obvious calculation. The other substeps are the four premisses of an instance of WF2 to which $\langle 5\rangle 1$ is the conclusion.

$\langle 6\rangle 1.$ $\land\ p \in Proc \land \overline{Enabled\langle\, CC(cc, cin, cout).CacheUpdate(p)\,\rangle_{\langle\, cc,cin,cout,ch\,\rangle}}$
$\equiv p \in Proc \land \exists\, c \in 1\,..\,Len(in[i])\ :\ NotRefresh_p(c)$
$\land\ p \in Proc \land Enabled\langle\, HCacheUpdate(p)\,\rangle_{hvars} \equiv p \in Proc \land NotRefresh_p(1)$
PROOF: This follows by direct calculation. □

Now, the first three premisses of WF2.

$\langle 6\rangle 2.$ $\land\ p \in Proc$
$\land\ \langle\, NotRefresh_p(1) \land HCacheUpdate(p)\,\rangle_{hvars}$
$\Rightarrow \langle\, \overline{CC(cc, cin, cout).CacheUpdate(p)}\,\rangle_{\overline{\langle\, cc,cin,cout,ch\,\rangle}}$
PROOF: The substance of this was shown already in the step simulation in $\langle 3\rangle 7$, and the variable change follows from $\langle 4\rangle 1$. □

$\langle 6\rangle 3.$ $\land\ p \in Proc$
$\land\ NotRefresh_p(1)$
$\land\ \langle\, HCacheUpdate(p)\,\rangle_{hvars}$
$\land\ \overline{Enabled\langle\, CC(cc, cin, cout).CacheUpdate(p)\,\rangle_{\langle\, cc,cin,cout,ch\,\rangle}}$
$\Rightarrow \land\ \langle\, HCacheUpdate(p)\,\rangle_{hvars}$
$\land\ p \in Proc$
$\land\ NotRefresh_p(1)$
PROOF: Immediate by propositional logic. □

$\langle 6\rangle 4.$ $\land\ NotRefresh_p(1)$
$\land\ \overline{Enabled\langle\, CC(cc, cin, cout).CacheUpdate(p)\,\rangle_{\langle\, cc,cin,cout,ch\,\rangle}}$
$\Rightarrow Enabled\langle\, HCacheUpdate(p)\,\rangle_{hvars}$
PROOF: Immediate by $\langle 6\rangle 1$ and propositional logic. □

The fourth premiss of WF2 is now proved, first by a manipulation in temporal logic, and then as the conclusion of an instance of the Lattice Rule.

$\langle 6\rangle 5.$ $\land\ \Box[\exists\, i \in Proc\ :\ HOp(i) \land \neg(\land\ p \in Proc$
$\land\ HCacheUpdate(p)$
$\land\ NotRefresh_p(1))]_{hvars}$
$\land\ \mathrm{WF}_{hvars}(HCacheUpdate_p)$
$\land\ \Diamond\Box\overline{Enabled\langle\, CC(cc, cin, cout).CacheUpdate(p)\,\rangle_{\langle\, cc,cin,cout,ch\,\rangle}}$
$\Rightarrow \Diamond\Box(p \in Proc \land NotRefresh_p(1))$
PROOF:

We first reduce the consequent by temporal logic, then consider it as the conclusion of the Lattice Rule to reduce it further to the Lattice Rule premiss which we prove using WF1. The *Bridge Rule* from [Lad95] is used to obtain $\langle 6\rangle 5$ from the conclusion of the Lattice Rule below. The Bridge Rule is strictly not a TLA proof, since it is not a valid TLA statement. It is rather a meta-statement that a certain rule of proof is a valid derived TLA inference rule. It's thus technically a meta-step, not a proof step. I argued in [Lad95] that such meta-steps were appropriate in a TLA proof.

First, the statement of the Bridge Rule.

LET: X, Y, Z : PROPOSITIONAL VARIABLE

$\langle 7\rangle 1.$ $$\frac{\Box X \land Y \Rightarrow \Box Y \qquad \Box X \Rightarrow \Box(Z \Rightarrow \Diamond Y)}{\Box X \land \Diamond\Box Z \Rightarrow \Diamond\Box Y}$$

PROOF SKETCH: See [Lad95, Page 46] for the proof. □

We prove the first Bridge-rule premiss, that NotRefresh is invariant under the hypotheses. This uses TLA rule INV1.

$\langle 7\rangle 2.$ $\land\ \Box[\exists\, i \in \mathit{Proc} : \mathit{HOp}(i) \land \neg(\land\ p \in \mathit{Proc}$
$\quad \land\ \mathit{HCacheUpdate}(p)$
$\quad \land\ \mathit{NotRefresh}_p(1))]_{\mathit{hvars}}$
$\land\ \mathrm{WF}_{\mathit{hvars}}(\mathit{HCacheUpdate}_p)$
$\land\ (p \in \mathit{Proc} \land \mathit{NotRefresh}_p(1))$
$\Rightarrow$
$\Box(p \in \mathit{Proc} \land \mathit{NotRefresh}_p(1))$

PROOF:

$\langle 8\rangle 1.$ $\land\ [\exists\, i \in \mathit{Proc} : \mathit{HOp}(i) \land \neg(\land\ p \in \mathit{Proc}$
$\quad \land\ \mathit{HCacheUpdate}(p)$
$\quad \land\ \mathit{NotRefresh}_p(1))]_{\mathit{hvars}}$
$\land\ (p \in \mathit{Proc} \land \mathit{NotRefresh}_p(1))$
$\Rightarrow$
$(p \in \mathit{Proc} \land \mathit{NotRefresh}'_p(1))$

PROOF: Immediate from the definitions of *HOp*, since *HCacheUpdate* is the only operation which changes $hin[p][1]$, and it follows by trivial propositional logic from the hypotheses that *HCacheUpdate* is false. Hence $hin[p][1]$ is UNCHANGED . □

$\langle 8\rangle 2.$ Q.E.D.

PROOF: Immediate by INV1 from $\langle 8\rangle 1$, and then strengthening the hypothesis. □

$\langle 7\rangle 3.$ $\land\ \Box[\exists\, i \in \mathit{Proc} : \mathit{HOp}(i) \land \neg(\land\ p \in \mathit{Proc}$
$\quad \land\ \mathit{HCacheUpdate}(p)$
$\quad \land\ \mathit{NotRefresh}_p(1))]_{\mathit{hvars}}$
$\land\ \mathrm{WF}_{\mathit{hvars}}(\mathit{HCacheUpdate}_p)$
$\Rightarrow$
$\overline{\mathit{Enabled}\langle\, \mathit{CC}(cc, cin, cout).\mathit{CacheUpdate}(p)\,\rangle_{\langle cc, cin, cout, ch\,\rangle}} \leadsto (p \in \mathit{Proc} \land \mathit{NotRefresh}_p(1))$

PROOF:

The substep is the premiss of the Lattice Rule.

LET: c : CONSTANT

$\langle 8\rangle 1.$ $\land\ \Box[\exists\, i \in \mathit{Proc} : \mathit{HOp}(i) \land \neg(\land\ p \in \mathit{Proc}$
$\quad \land\ \mathit{HCacheUpdate}(p)$
$\quad \land\ \mathit{NotRefresh}_p(1))]_{\mathit{hvars}}$
$\land\ \mathrm{WF}_{\mathit{hvars}}(\mathit{HCacheUpdate}_p)$
$\land\ c \in 1 \mathinner{..} \mathit{Len}(in[i])$
$\Rightarrow$
$\mathit{NotRefresh}_p(c) \leadsto \lor\ \exists\, d \in 1 \mathinner{..} \mathit{Len}(in[i]) : d < c \land \mathit{NotRefresh}_p(d)$
$\quad \lor\ p \in \mathit{Proc} \land \mathit{NotRefresh}_p(1)$

PROOF:

The three substeps are the three premises of an instance of WF1 to which $\langle 8\rangle 1$ is the conclusion.

$\langle 9\rangle 1.$ $\land\ p \in Proc$
$\land\ NotRefresh_p(c)$
$\land\ [\exists\, i \in Proc : HOp(i) \land \neg(\land\ p \in Proc$
$\land\ HCacheUpdate(p)$
$\land\ NotRefresh_p(1))]_{hvars}$
$\Rightarrow p \in Proc \land \lor\ NotRefresh'_p(c)$
$\lor\ \exists\, d \in 1 \,..\, Len(in'[i]) : d < c \land NotRefresh'_p(d)$
$\lor\ NotRefresh'_p(1)$

Proof:

$\langle 10\rangle 1.$ $\land\ p \in Proc$
$\land\ NotRefresh_p(c)$
$\land\ [\exists\, i \in Proc : HOp(i) \land \neg HCacheUpdate(p)]_{hvars}$
$\Rightarrow p \in Proc \land NotRefresh'_p(c)$

Proof: Immediate from definitions of the operations, and results about $HCacheUpdate$ in $\langle 3\rangle 7$. □

$\langle 10\rangle 2.$ $\land\ p \in Proc$
$\land\ NotRefresh_p(c)$
$\land\ \neg HCacheUpdate(p)$
$\land\ \neg NotRefresh_p(1)$
$\Rightarrow \land\ c \in 2 \,..\, Len(in[i])$
$\land\ NotRefresh'_p(c-1)$

Proof: Immediate from propositional logic, and results about $HCacheUpdate$ in $\langle 3\rangle 7$. □

$\langle 10\rangle 3.$ Q.E.D.

Proof: Follows by propositional logic from $\langle 10\rangle 1$ and $\langle 10\rangle 2$. □

$\langle 9\rangle 2.$ $NotRefresh_p(c)$
$\langle \land\ \exists\, i \in Proc : HOp(i)$
$\land\ \neg(\land\ p \in Proc$
$\land\ HCacheUpdate(p)$
$\land\ NotRefresh_p(1))$
$\land\ HCacheUpdate(p)\rangle_{hvars}$
$\Rightarrow \lor\ \exists\, d \in 1 \,..\, Len(in'[i]) : d < c \land NotRefresh'_p(d)$
$\lor\ p \in Proc \land NotRefresh'_p(1)$

Proof:

First, we reduce the premiss.

$\langle 10\rangle 1.$ $\land\ NotRefresh_p(c)$
$\land\ p \in Proc$
$\land\ HCacheUpdate(p)$
$\land\ \neg NotRefresh_p(1)$
$\land\ \neg$UNCHANGED $hvars$
$\Rightarrow \lor\ \exists\, d \in 1 \,..\, Len(in'[i]) : d < c \land NotRefresh'_p(d)$
$\lor\ p \in Proc \land NotRefresh'_p(1)$

Proof:

Then, everything follows easily by what's known for $HCacheUpdate$ from $\langle 3\rangle 7$.

$\langle 11\rangle 1.$ $\land\ NotRefresh_p(c)$
$\land\ p \in Proc$
$\land\ HCacheUpdate(p)$
$\land\ \neg NotRefresh_p(1)$
$\land\ \neg$UNCHANGED $hvars$
$\Rightarrow \land\ p \in Proc$
$\land\ NotRefresh_p(c)$
$\land\ c \in 2 \,..\, Len(in[i])$
$\land\ HCacheUpdate(p)$

Proof: Immediate by propositional logic and quantifier-case-splitting. □

⟨11⟩2. $\wedge\ p \in Proc \qquad \Rightarrow NotRefresh'_p(c-1)$
$\wedge\ NotRefresh_p(c)$
$\wedge\ c \in 2\,..\,Len(in[i])$
$\wedge\ HCacheUpdate(p)$

PROOF: Immediate from results in ⟨3⟩7. □

⟨11⟩3. Q.E.D.

PROOF: Immediate from ⟨11⟩1 and ⟨11⟩2 and existential generalisation over $c-1$ and propositional logic. □

⟨10⟩2. Q.E.D.

PROOF: Immediate by propositional logic. □

⟨9⟩3. $p \in Proc \wedge NotRefresh_p(c) \Rightarrow p \in Proc \wedge \exists\, d \in 1\,..\,Len(in[i])\ :\ NotRefresh_p(d)$

PROOF: Immediate by existential generalisation. □

⟨9⟩4. Q.E.D.

PROOF: (Almost) immediate from WF1 with the obvious substitution. Observe that $c \in 1\,..\,Len(in[p])$ is a conjunct of $NotRefresh_P(c)$ which is a conjunct of the consequent. Thus we have added $c \in 1\,..\,Len(in[p])$ to the hypotheses and retained an equivalent formula to the conclusion of WF2 from ⟨9⟩1, ⟨9⟩2 and ⟨9⟩3. □

⟨8⟩2. Q.E.D.

PROOF: Follows immediately from ⟨8⟩1 by the Lattice Rule with

$G \ \triangleq\ NotRefresh_p(1)$

$H_c \ \triangleq\ NotRefresh_p(c)$

$\succ\ \triangleq\ >,\ S\ \triangleq\ 1\,..\,Len(in[p]),\ F\ \triangleq$ *the other hypotheses* and using ⟨6⟩1. □

⟨7⟩4. Q.E.D.

PROOF: Follows by the Bridge Rule, ⟨7⟩1, from ⟨7⟩2 and ⟨7⟩3 with substitutions

$$\begin{aligned}\Box X \ \triangleq\ & \wedge\ \Box[\exists\, i \in Proc\ :\ HOp(i) \wedge \neg(\wedge\ p \in Proc \\ & \qquad\qquad\qquad\qquad\qquad\quad \wedge\ HCacheUpdate(p) \\ & \qquad\qquad\qquad\qquad\qquad\quad \wedge\ NotRefresh_p(1))]_{hvars} \\ & \wedge\ \mathrm{WF}_{hvars}(HCacheUpdate_p)\end{aligned}$$

$Y \ \triangleq\ (p \in Proc \wedge NotRefresh_p(1))$

$Z \ \triangleq\ \overline{Enabled\langle\, CC(cc, cin, cout).CacheUpdate(p)\,\rangle_{\langle\, cc, cin, cout, ch\,\rangle}}$. □

⟨6⟩6. Q.E.D.

PROOF: For fixed p, immediate by TLA rule WF2 from ⟨6⟩2, ⟨6⟩3, and ⟨6⟩4 with

$\mathcal{A} \ \triangleq\ HCacheUpdate(p)$,

$f \ \triangleq\ hvars$,

$\mathcal{B} \ \triangleq\ p \in Proc \wedge \mathcal{A} \wedge NotRefresh_p(1)$,

$\mathcal{M} \ \triangleq\ CC(cc, cin, cout, ch).CacheUpdate(p)$,

$g \ \triangleq\ \langle\, cc, cin, cout, ch\,\rangle$,

$P \ \triangleq\ p \in Proc \wedge NotRefresh_p(1)$. □

⟨5⟩2. Q.E.D.

PROOF: Immediate by quantification over p from ⟨5⟩1. □

⟨4⟩4. Q.E.D.

PROOF: Immediate by propositional logic from ⟨4⟩2 and ⟨4⟩3. □

⟨3⟩9. $\wedge\ HInit$
$\wedge\ \Box[\exists\, i \in Proc\ :\ HOp(i)]_{hvars}$
$\wedge\ Live$
$\Rightarrow \wedge\ \overline{CC(cc, cin, cout).Init}$
$\quad\ \wedge\ \Box[\overline{\exists\, i \in Proc\ :\ CC(cc, cin, cout).Next(i)}]_{\overline{\langle cc, cin, cout, ch\rangle}}$
$\quad\ \wedge\ \overline{Live}$

PROOF: ⟨3⟩9 follows from ⟨3⟩4, ⟨3⟩5, ⟨3⟩7 and ⟨3⟩8 by propositional logic. □

⟨3⟩10. Q.E.D.

The theorem follows by existential quantification over the free variables in ⟨3⟩9. □

Acknowledgements

Many thanks (and honorary membership of the Schadenfreude Society of Palo Alto) to Leslie Lamport, for showing me how to do TLA proofs and insisting I practice on this one. Thankyou also to Denis Roegel for reviewing my first versions of this proof and suggested necessary changes.

References

[ABM93] Yehuda Afek, Geoffrey Brown, and Michael Merritt. Lazy caching. *ACM Transactions on Programming Languages and Systems*, 15(1):182–205, January 1993.

[Ger96] Rob Gerth. Introduction to sequential consistency and the lazy caching algorithm. *Distributed Computing*, 1996. To appear.

[Lad95] Peter B. Ladkin. Formal but lively buffers in TLA+. 1995. Submitted for publication.

[Lam94] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[LLRO96] Peter Ladkin, Leslie Lamport, Denis Roegel, and Brian Olivier. Lazy caching in TLA. *Distributed Computing*, 1996. To appear.