A Reduction Theorem for Store Buffers

Ernie Cohen[1], Norbert Schirmer[2,⋆]

[1] Microsoft Corp., Redmond, WA, USA
[2] German Research Center for Artificial Intelligence (DFKI) Saarbrücken, Germany
ecohen@amazon.com, norbert.schirmer@web.de

Abstract. When verifying a concurrent program, it is usual to assume that memory is sequentially consistent. However, most modern multiprocessors depend on store buffering for efficiency, and provide native sequential consistency only at a substantial performance penalty. To regain sequential consistency, a programmer has to follow an appropriate programming discipline. However, naïve disciplines, such as protecting all shared accesses with locks, are not flexible enough for building high-performance multiprocessor software.

We present a new discipline for concurrent programming under TSO (total store order, with store buffer forwarding). It does not depend on concurrency primitives, such as locks. Instead, threads use ghost operations to acquire and release ownership of memory addresses. A thread can write to an address only if no other thread owns it, and can read from an address only if it owns it or it is shared and the thread has flushed its store buffer since it last wrote to an address it did not own. This discipline covers both coarse-grained concurrency (where data is protected by locks) as well as fine-grained concurrency (where atomic operations race to memory).

We formalize this discipline in Isabelle/HOL, and prove that if every execution of a program in a system without store buffers follows the discipline, then every execution of the program with store buffers is sequentially consistent. Thus, we can show sequential consistency under TSO by ordinary assertional reasoning about the program, without having to consider store buffers at all.

⋆ Work funded by the German Federal Ministry of Education and Research (BMBF) in the framework of the Verisoft XT project under grant 01 IS 07 008.

Table of Contents

1 Introduction

When verifying a shared-memory concurrent program, it is usual to assume that each memory operation works directly on a shared memory state, a model sometimes called *atomic* memory. A memory implementation that provides this abstraction for programs that communicate only through shared memory is said to be *sequentially consistent*. Concurrent algorithms in the computing literature tacitly assume sequential consistency, as do most application programmers.

However, modern computing platforms typically do not guarantee sequential consistency for arbitrary programs, for two reasons. First, optimizing compilers are typically incorrect unless the program is appropriately annotated to indicate which program locations might be concurrently accessed by other threads; this issue is addressed only cursorily in this report. Second, modern processors buffer stores of retired instructions. To make such buffering transparent to single-processor programs, subsequent reads of the processor read from these buffers in preference to the cache. (Otherwise, a program could write a new value to an address but later read an older value.) However, in a multiprocessor system, processors do not snoop the store buffers of other processors, so a store is visible to the storing processor before it is visible to other processors. This can result in executions that are not sequentially consistent.

The simplest example illustrating such an inconsistency is the following program, consisting of two threads T0 and T1, where `x` and `y` are shared memory variables (initially 0) and `r0` and `r1` are registers:

```
T0          T1

x = 1;      y = 1;
r0 = y;     r1 = x;
```

In a sequentially consistent execution, it is impossible for both `r0` and `r1` to be assigned 0. This is because the assignments to `x` and `y` must be executed in some order; if `x` (resp. `y`) is assigned first, then `r1` (resp. `r0`) will be set to 1. However, in the presence of store buffers, the assignments to `r0` and `r1` might be performed while the writes to `x` and `y` are still in their respective store buffers, resulting in both `r0` and `r1` being assigned 0.

One way to cope with store buffers is make them an explicit part of the programming model. However, this is a substantial programming concession. First, because store buffers are FIFO, it ratchets up the complexity of program reasoning considerably; for example, the reachability problem for a finite set of concurrent finite-state programs over a finite set of finite-valued locations is in PSPACE without store buffers, but undecidable (even for two threads) with store buffers. Second, because writes from function calls might still be buffered when a function returns, making the store buffers explicit would break modular program reasoning.

In practice, the usual remedy for store buffering is adherence to a programming discipline that provides sequential consistency for a suitable class of architectures. In this report, we describe and prove the correctness of such a discipline suitable for the memory model provided by existing x86/x64 machines, where each write emerging from a store buffer hits a global cache visible to all processors. Because each processor sees the same global ordering of writes, this model is sometimes called *total store order* (TSO) [2][3]

The concurrency discipline most familiar to concurrent programs is one where each variable is protected by a lock, and a thread must hold the corresponding lock to access the variable. (It is possible to generalize this to allow shared locks, as well as variants such as split semaphores.) Such lock-based techniques are typically referred to as *coarse-grained* concurrency control, and suffice for most concurrent application programming. However, these techniques do not suffice for low-level system programming (e.g., the construction of OS kernels), for several reasons. First, in kernel programming efficiency is paramount, and atomic memory operations are more efficient for many problems. Second, lock-free concurrency control can sometimes guarantee stronger correctness (e.g., wait-free algorithms can provide bounds on execution time). Third, kernel programming requires taking into account the implicit concurrency of concurrent hardware activities (e.g., a hardware TLB racing to use page tables while the kernel is trying to access them), and hardware cannot be forced to follow a locking discipline.

A more refined concurrency control discipline, one that is much closer to expert practice, is to classify memory addresses as lock-protected or shared. Lock-protected addresses are used in the usual way, but shared addresses can be accessed using atomic operations provided by hardware (e.g., on x86 class architectures, most reads and writes are atomic[4]). The main restriction on these accesses is that if a processor does a shared write and a

[3] Before 2008, Intel [9] and AMD [1] both put forward a weaker memory model in which writes to different memory addresses may be seen in different orders on different processors, but respecting causal ordering. However, current implementations satisfy the stronger conditions described in this report and are also compliant with the latest revisions of the Intel specifications [10]. According to Owens et al. [15] AMD is also planning a similar adaptation of their manuals.

[4] This atomicity isn't guaranteed for certain memory types, or for operations that cross a cache line.

subsequent shared read (possibly from a different address), the processor must flush the store buffer somewhere in between. For example, in the example above, both `x` and `y` would be shared addresses, so each processor would have to flush its store buffer between its first and second operations.

However, even this discipline is not very satisfactory. First, we would need even more rules to allow locks to be created or destroyed, or to change memory between shared and protected, and so on. Second, there are many interesting concurrency control primitives, and many algorithms, that allow a thread to obtain exclusive ownership of a memory address; why should we treat locking as special?

In this report, we consider a much more general and powerful discipline that also guarantees sequential consistency. The basic rule for shared addresses is similar to the discipline above, but there are no locking primitives. Instead, we treat *ownership* as fundamental. The difference is that ownership is manipulated by nonblocking ghost updates, rather than an operation like locking that have runtime overhead. Informally the rules of the discipline are as follows:

- In any state, each memory address is either *shared* or *unshared*. Each memory address is also either *owned* by a unique thread or *unowned*. Every unowned address must be shared. Each address is also either read-only or read-write. Every read-only address is unowned.
- A thread can (autonomously) acquire ownership of an unowned address, or release ownership of a address that it owns. It can also change whether an address it owns is shared or not. Upon release of an address it can mark it as read-only.
- Each memory access is marked as *volatile* or *non-volatile*.
- A thread can perform a write if it is *sound*. It can perform a read if it is sound and *clean*.
- A non-volatile write is sound if the thread owns the address and the address is unshared.
- A non-volatile read is sound if the thread owns the address or the address is read-only.
- A volatile write is sound if no other thread owns the address and the address is not marked as read-only.
- A volatile read is sound if the address is shared or the thread owns it.
- A volatile read is clean if the store buffer has been flushed since the last volatile write. Moreover, every non-volatile read is clean.
- For interlocked operations (like compare and swap), which have the side effect of the store buffer getting flushed, the rules for volatile accesses apply.

Note first that these conditions are not thread-local, because some actions are allowed only when an address is unowned, marked read-only, or not marked read-only. A thread can ascertain such conditions only through system-wide invariants, respected by all threads, along with data it reads. By imposing suitable global invariants, various thread-local disciplines (such as one where addresses are protected by locks, conditional critical reasons, or monitors) can be derived as lemmas by ordinary program reasoning, without need for meta-theory.

Second, note that these rules can be checked in the context of a concurrent program without store buffers, by introducing ghost state to keep track of ownership and sharing and whether the thread has performed a volatile write since the last flush. Our main result is that if a program obeys the rules above, then the program is sequentially consistent when executed on a TSO machine.

Consider our first example program. If we choose to leave both `x` and `y` unowned (and hence shared), then all accesses must be volatile. This would force each thread to flush the store buffer between their first and second operations. In practice, on an x86/x64 machine, this would be done by making the writes interlocked, which flushes store buffers as a side effect. Whichever thread flushes its store buffer second is guaranteed to see the write of the other thread, making the execution violating sequential consistency impossible.

However, couldn't the first thread try to take ownership of `x` before writing it, so that its write could be non-volatile? The answer is that it could, but then the second thread would be unable to read `x` volatile (or take ownership of `x` and read it non-volatile), because we would be unable to prove that `x` is unowned at that point. In other words, a thread can take ownership of an address only if it is not racing to do so.

Ultimately, the races allowed by the discipline involve volatile access to a shared address, which brings us back to locks. A spinlock is typically implemented with an interlocked read-modify-write on an address (the interlocking providing the required flushing of the store buffer). If the locking succeeds, we can prove (using for example a ghost variable giving the ID of the thread taking the lock) that no other thread holds the lock, and can therefore safely take ownership of an address "protected" by the lock (using the global invariant that only the lock owner can own the protected address). Thus, our discipline subsumes the better-known disciplines governing coarse-grained concurrency control.

To summarize, our motivations for using ownership as our core notion of a practical programming discipline are the following:

1. the distinction between global (volatile) and local (non-volatile) accesses is a practical requirement to reduce the performance penalty due to necessary flushes and to allow important compiler optimizations (such as moving a local write ahead of a global read),
2. coarse-grained concurrency control like locking is nothing special but only a derived concept which is used for ownership transfer (any other concurrency control that guarantees exclusive access is also fine), and
3. we want that the conditions to check for the programming discipline can be discharged by ordinary state-based program reasoning on a sequentially consistent memory model (without having to talk about histories or complete executions).

Overview In Section 2 we introduce preliminaries of Isabelle/HOL, the theorem prover in which we mechanized our work. In Section 3 we informally describe the programming discipline and basic ideas of the formalization, which is detailed in Section 4 where we introduce the formal models and the reduction theorem. In Section 5 we give some details of important building blocks for the proof of the reduction theorem. To illustrate the connection between a programming language semantics and our reduction theorem, we instantiate our framework with a simple semantics for a parallel WHILE language in Section 6. Finally we conclude in Section 7.

2 Preliminaries

The formalization presented in this papaer is mechanized and checked within the generic interactive theorem prover *Isabelle* [16]. Isabelle is called generic as it provides a framework to formalize various *object logics* declared via natural deduction style inference rules. The object logic that we employ for our formalization is the higher order logic of *Isabelle/HOL* [12].

This article is written using Isabelle's document generation facilities, which guarantees a close correspondence between the presentation and the actual theory files. We distinguish formal entities typographically from other text. We use a sans serif font for types and constants (including functions and predicates), e.g., map, a slanted serif font for free variables, e.g., x, and a slanted sans serif font for bound variables, e.g., x. Small capitals are used for data type constructors, e.g., FOO, and type variables have a leading tick, e.g., $'a$. HOL keywords are typeset in type-writer font, e.g., `let`.

To group common premises and to support modular reasoning Isabelle provides *locales* [4, 5]. A locale provides a name for a context of fixed parameters and premises, together with an elaborate infrastructure to define new locales by inheriting and extending other locales, prove theorems within locales and interpret (instantiate) locales. In our formalization we employ this infrastructure to separate the memory system from the programming language semantics.

The logical and mathematical notions follow the standard notational conventions with a bias towards functional programming. We only present the more unconventional parts here. We prefer curried function application, e.g., $f\ a\ b$ instead of $f(a,\ b)$. In this setting the latter becomes a function application to *one* argument, which happens to be a pair.

Isabelle/HOL provides a library of standard types like Booleans, natural numbers, integers, total functions, pairs, lists, and sets. Moreover, there are packages to define new data types and records. Isabelle allows polymorphic types, e.g., $'a$ list is the list type with type variable $'a$. In HOL all functions are total, e.g., $\mathsf{nat} \Rightarrow \mathsf{nat}$ is a total function on natural numbers. A function update is $f(y := v) = (\lambda x.\ \mathsf{if}\ x = y\ \mathsf{then}\ v\ \mathsf{else}\ f\ x)$. To formalize partial functions the type $'a$ option is used. It is a data type with two constructors, one to inject values of the base type, e.g., $\lfloor x \rfloor$, and the additional element $\bot$. A base value can be projected with the function the, which is defined by the sole equation $\mathsf{the}\ \lfloor x \rfloor = x$. Since HOL is a total logic the term $\mathsf{the}\ \bot$ is still a well-defined yet un(der)specified value. Partial functions are usually represented by the type $'a \Rightarrow 'b\ \mathsf{option}$, abbreviated as $'a \rightharpoonup 'b$. They are commonly used as *maps*. We denote the domain of map m by $\mathsf{dom}\ m$. A map update is written as $m(a \mapsto v)$. We can restrict the domain of a map m to a set A by $m\restriction_A$.

The syntax and the operations for lists are similar to functional programming languages like ML or Haskell. The empty list is [], with $x\ \#\ xs$ the element x is 'consed' to the list xs.With $xs\ @\ ys$ list ys is appended to list xs. With the term $\mathsf{map}\ f\ xs$ the function f is applied to all elements in xs. The length of a list is $|xs|$, the n-th element of a list can be selected with $xs_{[n]}$ and can be updated via $xs[n := v]$. With $\mathsf{dropWhile}\ P\ xs$ the prefix for which all elements satisfy predicate P are dropped from list xs.

Sets come along with the standard operations like union, i.e., $A \cup B$, membership, i.e., $x \in A$ and set inversion, i.e., $-\ A$.

Tuples with more than two components are pairs nested to the right.

3 Programming discipline

For sequential code on a single processor the store buffer is invisible, since reads respect outstanding writes in the buffer. This argument can be extended to thread local memory in the context of a multiprocessor architecture. Memory typically becomes temporarily thread local by means of locking. The C-idiom to identify shared portions of the memory is the `volatile` tag on variables and type declarations. Thread local memory can be accessed non-volatilely, whereas accesses to shared memory are tagged as volatile. This prevents the compiler from applying certain optimizations to those accesses which could cause undesired behavior, e.g., to store intermediate values in registers instead of writing them to the memory.

The basic idea behind the programming discipline is, that before gathering new information about the shared state (via reading) the thread has to make its outstanding changes to the shared state visible to others (by flushing the store buffer). This allows to sequentialize the reads and writes to obtain a sequentially consistent execution of the global system. In this sequentialization a write to shared memory happens when the write instruction exits the store buffer, and a read from the shared memory happens when all preceding writes have exited.

We distinguish thread local and shared memory by an ownership model. Ownership is maintained in ghost state and can be transferred as side effect of write operations and by a dedicated ghost operation. Every thread has a set of owned addresses. Owned addresses of different threads are disjoint. Moreover, there is a global set of shared addresses which can additionally be marked as read-only. Unowned addresses — addresses owned by no thread — can be accessed concurrently by all threads. They are a subset of the shared addresses. The read-only addresses are a subset of the unowned addresses (and thus of the shared addresses). We only allow a thread to write to owned addresses and unowned, read-write addresses. We only allow a thread to read from owned addresses and from shared addresses (even if they are owned by another thread).

All writes to shared memory have to be volatile. Reads from shared addresses also have to be volatile, except if the address is owned (i.e., single writer, multiple readers) or if the address is read-only. Moreover, non-volatile writes are restricted to owned, unshared memory. As long as a thread owns an address it is guaranteed that it is the only one writing to that address. Hence this thread can safely perform non-volatile reads to that address without missing any write. Similar it is safe for any thread to access read-only memory via non-volatile reads since there are no outstanding writes at all.

Recall that a volatile read is *clean* if it is guaranteed that there is no outstanding volatile write (to any address) in the store buffer. Moreover every non-volatile read is clean. To regain sequential consistency under the presence of store buffers every thread has to make sure that every read is clean, by flushing the store buffer when necessary. To check the flushing policy of a thread, we keep track of clean reads by means of ghost state. For every thread we maintain a dirty flag. It is reset as the store buffer gets flushed. Upon a volatile write the dirty flag is set. The dirty flag is considered to guarantee that a volatile read is clean.

Table 1a summarizes the access policy and Table 1b the associated flushing policy of the programming discipline. The key motivation is to improve performance by minimizing the number of store buffer flushes, while staying sequentially consistent. The need for flushing the store buffer decreases from interlocked accesses (where flushing is a side-effect) over volatile accesses to non-volatile accesses. From the viewpoint of access rights there is no difference between interlocked and volatile accesses. However, keep in mind that some interlocked operations can read from, modify and write to an address in a single atomic step of the underlying hardware and are typically used in lock-free algorithms or for the implementation of locks.

Table 1: Programming discipline.

(a) Access policy

	shared (read-write)	shared (read-only)	unshared
un-owned	vR, vW	vR, R	unreachable
owned	vR, vW, R	unreachable	vR, vW, R, W
owned by other	vR	unreachable	

(v)olatile, (R)ead, (W)rite
all reads have to be clean

(b) Flushing policy

	flush (before)
interlocked	as side effect
vR	if not clean
R, vW, W	never

4 Formalization

In this section we go into the details of our formalization. In our model, we distinguish the plain ‘memory system’ from the ‘programming language semantics’ which we both describe as a small-step transition relation. During a computation the programming language issues memory instructions (read / write) to the memory system, which itself returns the results in temporary registers. This clean interface allows us to parameterize the program semantics over the memory system. Our main theorem allows us to simulate a computation step in the semantics based on a memory system with store buffers by n steps in the semantics based on a sequentially consistent memory system. We refer to the former one as *store buffer machine* and to the latter one as *virtual machine*. The simulation theorem is independent of the programming language.

We continue with introducing the common parts of both machines. In Section 4.1 we describe the store buffer machine and in Section 4.2 we then describe the virtual machine. The main reduction theorem is presented in 4.3.

Addresses a, values v and temporaries t are natural numbers. Ghost annotations for manipulating the ownership information are the following sets of addresses: the acquired addresses A, the unshared (local) fraction L of the acquired addresses, the released addresses R and the writable fraction W of the released addresses (the remaining addresses are considered read-only). These ownership annotations are considered as side-effects on volatile writes and interlocked operations (in case a write is performed). Moreover, a special ghost instruction allows to transfer ownership. The possible status changes of an address due to these ownership transfer operations are depicted in Figure 1. Note that ownership of an address is not directly transferred between threads, but is first released by one thread and then can be acquired by another thread. A memory instruction is a datatype with the

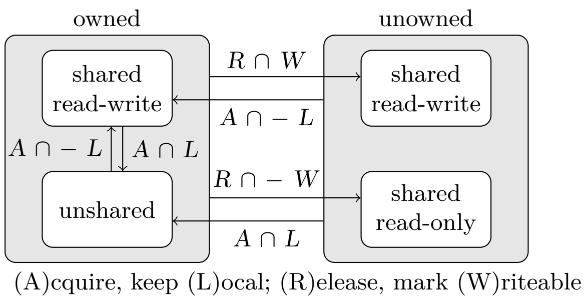

Fig. 1: Ownership transfer

following constructors:

- READ *volatile* a t for reading from address a to temporary t, where the Boolean *volatile* determines whether the access is volatile or not.
- WRITE *volatile* a *sop* A L R W to write the result of evaluating the store operation *sop* at address a. A store operation is a pair (D, f), with the domain D and the function f. The function f takes temporaries j as a parameter, which maps a temporary to a value. The subset of temporaries that is considered by function f is specified by the domain D. We consider store operations as valid when they only depend on their domain:

  $\textsf{valid-sop}\ sop \equiv \forall D\ f\ j.\ sop = (D, f) \wedge D \subseteq \textsf{dom}\ j \longrightarrow f\ j = f\ (j{\restriction}_D)$

Again the Boolean *volatile* specifies the kind of memory access.

- RMW a t *sop cond ret* A L R W, for atomic interlocked 'read-modify-write' instructions (flushing the store buffer). First the value at address a is loaded to temporary t, and then the condition *cond* on the temporaries is considered to decide whether a store operation is also executed. In case of a store the function *ret*, depending on both the old value at address a and the new value (according to store operation *sop*), specifies the final result stored in temporary t. With a trivial condition *cond* this instruction also covers interlocked reads and writes.
- FENCE, a memory fence that flushes the store buffer.
- GHOST A L R W for ownership transfer.

4.1 Store buffer machine

For the store buffer machine the configuration of a single thread is a tuple (p, is, j, sb) consisting of the program state p, a memory instruction list is, the map of temporaries j and the store buffer sb. A global configuration of the store buffer machine (ts, m) consists of a list of thread configurations ts and the memory m, which is a function from addresses to values.

We describe the computation of the global system by the non-deterministic transition relation $(ts, m) \overset{\mathsf{sb}}{\Rightarrow} (ts', m')$ defined in Figure 2.

$$\frac{i < |ts| \qquad ts_{[i]} = (p, is, j, sb) \qquad j \vdash p \rightarrow_{\mathsf{p}} (p', is')}{(ts, m) \overset{\mathsf{sb}}{\Rightarrow} (ts[i := (p', is \mathbin{@} is', j, sb)], m)}$$

$$\frac{i < |ts| \qquad ts_{[i]} = (p, is, j, sb) \qquad (is, j, sb, m) \overset{\mathsf{sb}}{\rightarrow}_{\mathsf{m}} (is', j', sb', m')}{(ts, m) \overset{\mathsf{sb}}{\Rightarrow} (ts[i := (p, is', j', sb')], m')}$$

$$\frac{i < |ts| \qquad ts_{[i]} = (p, is, j, sb) \qquad (m, sb) \rightarrow_{\mathsf{sb}} (m', sb')}{(ts, m) \overset{\mathsf{sb}}{\Rightarrow} (ts[i := (p, is, j, sb')], m')}$$

Fig. 2: Global transitions of store buffer machine

A transition selects a thread $ts_{[i]} = (p, is, j, sb)$ and either the 'program' the 'memory' or the 'store buffer' makes a step defined by sub-relations.

The program step relation is a parameter to the global transition relation. A program step $j \vdash p \rightarrow_{\mathsf{p}} (p', is')$ takes the temporaries j and the current program state p and makes a step by returning a new program state p' and an instruction list is' which is appended to the remaining instructions.

A memory step $(is, j, sb, m) \overset{\mathsf{sb}}{\rightarrow}_{\mathsf{m}} (is', j', sb', m')$ of a machine with store buffer may only fill its store buffer with new writes.

In a store buffer step $(m, sb) \rightarrow_{\mathsf{sb}} (m', sb')$ the store buffer may release outstanding writes to the memory.

The store buffer maintains the list of outstanding memory writes. Write instructions are appended to the end of the store buffer and emerge to memory from the front of the list. A read instructions is satisfied from the store buffer if possible. An entry in the store buffer is of the form $\textsc{Write}_{\mathsf{sb}}$ *volatile* a *sop* v for an outstanding write (keeping the volatile flag), where operation *sop* evaluated to value v.

As defined in Figure 3 a write updates the memory when it exits the store buffer.

$$\overline{(m,\ \textsc{Write}_{\mathsf{sb}}\ \mathit{volatile}\ a\ \mathit{sop}\ v\ A\ L\ R\ W\ \#\ \mathit{sb}) \rightarrow_{\mathsf{sb}} (m(a := v),\ \mathit{sb})}$$

Fig. 3: Store buffer transition

$$\frac{v = (\mathtt{case}\ \mathsf{buffered\text{-}val}\ \mathit{sb}\ a\ \mathtt{of}\ \bot \Rightarrow m\ a \mid \lfloor v' \rfloor \Rightarrow v')}{(\textsc{Read}\ \mathit{volatile}\ a\ t\ \#\ \mathit{is},\ j,\ \mathit{sb},\ m) \xrightarrow{\mathsf{sb}}_{\mathsf{m}} (\mathit{is},\ j(t \mapsto v),\ \mathit{sb},\ m)}$$

$$\frac{\mathit{sb}' = \mathit{sb}\ @\ [\textsc{Write}_{\mathsf{sb}}\ \mathit{volatile}\ a\ (D,\ f)\ (f\ j)\ A\ L\ R\ W]}{(\textsc{Write}\ \mathit{volatile}\ a\ (D,\ f)\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ \mathit{sb},\ m) \xrightarrow{\mathsf{sb}}_{\mathsf{m}} (\mathit{is},\ j,\ \mathit{sb}',\ m)}$$

$$\frac{\neg\ \mathit{cond}\ (j(t \mapsto m\ a)) \qquad j' = j(t \mapsto m\ a)}{(\mathrm{RMW}\ a\ t\ (D,\ f)\ \mathit{cond}\ \mathit{ret}\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ [],\ m) \xrightarrow{\mathsf{sb}}_{\mathsf{m}} (\mathit{is},\ j',\ [],\ m)}$$

$$\frac{\mathit{cond}\ (j(t \mapsto m\ a)) \qquad j' = j(t \mapsto \mathit{ret}\ (m\ a)\ (f\ (j(t \mapsto m\ a)))) \qquad m' = m(a := f\ (j(t \mapsto m\ a)))}{(\mathrm{RMW}\ a\ t\ (D,\ f)\ \mathit{cond}\ \mathit{ret}\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ [],\ m) \xrightarrow{\mathsf{sb}}_{\mathsf{m}} (\mathit{is},\ j',\ [],\ m')}$$

$$\overline{(\textsc{Fence}\ \#\ \mathit{is},\ j,\ [],\ m) \xrightarrow{\mathsf{sb}}_{\mathsf{m}} (\mathit{is},\ j,\ [],\ m)}$$

$$\overline{(\textsc{Ghost}\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ \mathit{sb},\ m) \xrightarrow{\mathsf{sb}}_{\mathsf{m}} (\mathit{is},\ j,\ \mathit{sb},\ m)}$$

Fig. 4: Memory transitions of store buffer machine

The memory transition are defined in Figure 4. With buffered-val $\mathit{sb}\ a$ we obtain the value of the last write to address a which is still pending in the store buffer. In case no outstanding write is in the store buffer we read from the memory. Store operations have no immediate effect on the memory but are queued in the store buffer instead. Interlocked operations and the fence operation require an empty store buffer, which means that it has to be flushed before the action can take place. The read-modify-write instruction first adds the current value at address a to temporary t and then checks the store condition *cond* on the temporaries. If it fails this read is the final result of the operation. Otherwise the store is performed. The resulting value of the temporary t is specified by the function *ret* which considers both the old and new value as input. The fence and the ghost instruction are just skipped.

4.2 Virtual machine

The virtual machine is a sequentially consistent machine without store buffers, maintaining additional ghost state to check for the programming discipline. A thread configuration is a tuple $(p,\ \mathit{is},\ j,\ \mathcal{D},\ \mathcal{O})$, with a dirty flag $\mathcal{D}$ indicating whether there may be an outstanding volatile write in the store buffer and the set of owned addresses $\mathcal{O}$. The dirty flag $\mathcal{D}$ is considered to specify if a read is clean: for *all* volatile reads the dirty flag must not be set. The global configuration of the virtual machine $(\mathit{ts},\ m,\ \mathcal{S})$ maintains a Boolean map of shared addresses $\mathcal{S}$ (indicating write permission). Addresses in the domain of mapping $\mathcal{S}$ are considered shared and the set of read-only addresses is obtained from $\mathcal{S}$ by: read-only $\mathcal{S} \equiv \{a.\ \mathcal{S}\ a = \lfloor \mathsf{False} \rfloor\}$

According to the rules in Fig 5 a global transition of the virtual machine $(\mathit{ts},\ m,\ \mathcal{S}) \overset{\mathsf{v}}{\Rightarrow} (\mathit{ts}',\ m',\ \mathcal{S}')$ is either a program or a memory step. The transition rules for its memory system are defined in Figure 6. In addition to the transition rules for the virtual machine we introduce the *safety* judgment $\mathcal{O}s,i \vdash (\mathit{is},\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd$ in Figure 7, where $\mathcal{O}s$ is the list of ownership sets obtained from the thread list ts and i is the index of the current

$$\frac{i < |ts| \qquad ts_{[i]} = (p,\, is,\, j,\, \mathcal{D},\, \mathcal{O}) \qquad j \vdash p \to_{\mathsf{p}} (p',\, is')}{(ts,\, m,\, \mathcal{S}) \overset{\mathsf{v}}{\Rightarrow} (ts[i := (p',\, is \mathbin{@} is',\, j,\, \mathcal{D},\, \mathcal{O})],\, m,\, \mathcal{S})}$$

$$\frac{i < |ts| \qquad ts_{[i]} = (p,\, is,\, j,\, \mathcal{D},\, \mathcal{O}) \qquad (is,\, j,\, m,\, \mathcal{D},\, \mathcal{O},\, \mathcal{S}) \overset{\mathsf{v}}{\to}_{\mathsf{m}} (is',\, j',\, m',\, \mathcal{D}',\, \mathcal{O}',\, \mathcal{S}')}{(ts,\, m,\, \mathcal{S}) \overset{\mathsf{v}}{\Rightarrow} (ts[i := (p,\, is',\, j',\, \mathcal{D}',\, \mathcal{O}')],\, m',\, \mathcal{S}')}$$

Fig. 5: Global transitions of virtual machine

$$\frac{}{(\textsc{Read}\ \mathit{volatile}\ a\ t \mathbin{\#} is,\, j,\, x,\, m,\, \mathit{ghst}) \overset{\mathsf{v}}{\to}_{\mathsf{m}} (is,\, j(t \mapsto m\ a),\, x,\, m,\, \mathit{ghst})}$$

$$\frac{}{(\textsc{Write}\ \mathsf{False}\ a\ (D,\, f)\ A\ L\ R\ W \mathbin{\#} is,\, j,\, x,\, m,\, \mathit{ghst}) \overset{\mathsf{v}}{\to}_{\mathsf{m}} (is,\, j,\, x,\, m(a := f\ j),\, \mathit{ghst})}$$

$$\frac{\mathit{ghst} = (\mathcal{D},\, \mathcal{O},\, \mathcal{S}) \qquad \mathit{ghst}' = (\mathsf{True},\, \mathcal{O} \cup A - R,\, \mathcal{S} \oplus_{W} R \ominus_{A} L)}{(\textsc{Write}\ \mathsf{True}\ a\ (D,\, f)\ A\ L\ R\ W \mathbin{\#} is,\, j,\, x,\, m,\, \mathit{ghst}) \overset{\mathsf{v}}{\to}_{\mathsf{m}} (is,\, j,\, x,\, m(a := f\ j),\, \mathit{ghst}')}$$

$$\frac{\neg\ \mathit{cond}\ (j(t \mapsto m\ a)) \qquad \mathit{ghst} = (\mathcal{D},\, \mathcal{O},\, \mathcal{S}) \qquad \mathit{ghst}' = (\mathsf{False},\, \mathcal{O},\, \mathcal{S})}{(\mathrm{RMW}\ a\ t\ (D,\, f)\ \mathit{cond}\ \mathit{ret}\ A\ L\ R\ W \mathbin{\#} is,\, j,\, x,\, m,\, \mathit{ghst}) \overset{\mathsf{v}}{\to}_{\mathsf{m}} (is,\, j(t \mapsto m\ a),\, x,\, m,\, \mathit{ghst}')}$$

$$\frac{\begin{array}{c}\mathit{cond}\ (j(t \mapsto m\ a)) \qquad j' = j(t \mapsto \mathit{ret}\ (m\ a)\ (f\ (j(t \mapsto m\ a)))) \\ m' = m(a := f\ (j(t \mapsto m\ a))) \qquad \mathit{ghst} = (\mathcal{D},\, \mathcal{O},\, \mathcal{S}) \qquad \mathit{ghst}' = (\mathsf{False},\, \mathcal{O} \cup A - R,\, \mathcal{S} \oplus_{W} R \ominus_{A} L)\end{array}}{(\mathrm{RMW}\ a\ t\ (D,\, f)\ \mathit{cond}\ \mathit{ret}\ A\ L\ R\ W \mathbin{\#} is,\, j,\, x,\, m,\, \mathit{ghst}) \overset{\mathsf{v}}{\to}_{\mathsf{m}} (is,\, j',\, x,\, m',\, \mathit{ghst}')}$$

$$\frac{\mathit{ghst} = (\mathcal{D},\, \mathcal{O},\, \mathcal{S}) \qquad \mathit{ghst}' = (\mathsf{False},\, \mathcal{O},\, \mathcal{S})}{(\textsc{Fence} \mathbin{\#} is,\, j,\, x,\, m,\, \mathit{ghst}) \overset{\mathsf{v}}{\to}_{\mathsf{m}} (is,\, j,\, x,\, m,\, \mathit{ghst}')}$$

$$\frac{\mathit{ghst} = (\mathcal{D},\, \mathcal{O},\, \mathcal{S}) \qquad \mathit{ghst}' = (\mathcal{D},\, \mathcal{O} \cup A - R,\, \mathcal{S} \oplus_{W} R \ominus_{A} L)}{(\textsc{Ghost}\ A\ L\ R\ W \mathbin{\#} is,\, j,\, x,\, m,\, \mathit{ghst}) \overset{\mathsf{v}}{\to}_{\mathsf{m}} (is,\, j,\, x,\, m,\, \mathit{ghst}')}$$

Fig. 6: Memory transitions of the virtual machine

thread. Safety of all reachable states of the virtual machine ensures that the programming discipline is obeyed by the program and is our formal prerequisite for the simulation theorem. It is left as a proof obligation to be discharged by means of a proper program logic for sequentially consistent executions. In the following we elaborate on the rules of

$$\frac{a \in \mathcal{O} \vee a \in \textsf{read-only}\ \mathcal{S} \vee \mathit{volatile} \wedge a \in \textsf{dom}\ \mathcal{S} \qquad \mathit{volatile} \longrightarrow \neg\, \mathcal{D}}{\mathcal{O}s,i \vdash (\textsc{Read}\ \mathit{volatile}\ a\ t\ \#\ \mathit{is},\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd}$$

$$\frac{a \in \mathcal{O} \qquad a \notin \textsf{dom}\ \mathcal{S}}{\mathcal{O}s,i \vdash (\textsc{Write}\ \textsf{False}\ a\ (D,\ f)\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd}$$

$$\frac{\begin{array}{c}\forall j{<}|\mathcal{O}s|.\ i \neq j \longrightarrow a \notin \mathcal{O}s_{[j]} \qquad a \notin \textsf{read-only}\ \mathcal{S} \\ \forall j{<}|\mathcal{O}s|.\ i \neq j \longrightarrow A \cap \mathcal{O}s_{[j]} = \emptyset \qquad A \subseteq \mathcal{O} \cup \textsf{dom}\ \mathcal{S} \qquad L \subseteq A \qquad R \subseteq \mathcal{O} \qquad A \cap R = \emptyset\end{array}}{\mathcal{O}s,i \vdash (\textsc{Write}\ \textsf{True}\ a\ (D,\ f)\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd}$$

$$\frac{\neg\ \mathit{cond}\ (j(t \mapsto m\ a)) \qquad a \in \textsf{dom}\ \mathcal{S} \cup \mathcal{O}}{\mathcal{O}s,i \vdash (\text{RMW}\ a\ t\ (D,\ f)\ \mathit{cond}\ \mathit{ret}\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd}$$

$$\frac{\begin{array}{c}\mathit{cond}\ (j(t \mapsto m\ a)) \qquad \forall j{<}|\mathcal{O}s|.\ i \neq j \longrightarrow a \notin \mathcal{O}s_{[j]} \qquad a \notin \textsf{read-only}\ \mathcal{S} \\ \forall j{<}|\mathcal{O}s|.\ i \neq j \longrightarrow A \cap \mathcal{O}s_{[j]} = \emptyset \qquad A \subseteq \mathcal{O} \cup \textsf{dom}\ \mathcal{S} \qquad L \subseteq A \qquad R \subseteq \mathcal{O} \qquad A \cap R = \emptyset\end{array}}{\mathcal{O}s,i \vdash (\text{RMW}\ a\ t\ (D,\ f)\ \mathit{cond}\ \mathit{ret}\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd}$$

$$\frac{}{\mathcal{O}s,i \vdash (\textsc{Fence}\ \#\ \mathit{is},\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd}$$

$$\frac{A \subseteq \textsf{dom}\ \mathcal{S} \cup \mathcal{O} \qquad L \subseteq A \qquad R \subseteq \mathcal{O} \qquad A \cap R = \emptyset \qquad \forall j{<}|\mathcal{O}s|.\ i \neq j \longrightarrow A \cap \mathcal{O}s_{[j]} = \emptyset}{\mathcal{O}s,i \vdash (\textsc{Ghost}\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd}$$

Fig. 7: Safe configurations of a virtual machine

Figures 6 and 7 in parallel. To read from an address it either has to be owned or read-only or it has to be volatile and shared. Moreover the read has to be clean. The memory content of address a is stored in temporary t. Non-volatile writes are only allowed to owned and unshared addresses. The result is written directly into the memory. A volatile write is only allowed when no other thread owns the address and the address is not marked as read-only. Simultaneously with the volatile write we can transfer ownership as specified by the annotations A, L, R and W. The acquired addresses A must not be owned by any other thread and stem from the shared addresses or are already owned. Reacquiring owned addresses can be used to change the shared-status via the set of local addresses L which have to be a subset of A. The released addresses R have to be owned and distinct from the acquired addresses A. After the write the new ownership set of the thread is obtained by adding the acquired addresses A and releasing the addresses R: $\mathcal{O} \cup A - R$. The released addresses R are augmented to the shared addresses S and the local addresses L are removed. We also take care about the write permissions in the shared state: the released addresses in set W as well as the acquired addresses are marked writable: $\mathcal{S} \oplus_W R \ominus_A L$. The auxiliary ternary operators to augment and subtract addresses from the sharing map are defined as follows:

$\mathcal{S} \oplus_W R \equiv \lambda a.\ \textsf{if}\ a \in R\ \textsf{then}\ \lfloor a \in W \rfloor\ \textsf{else}\ \mathcal{S}\ a$

$\mathcal{S} \ominus_A L \equiv$
$\lambda a.\ \textsf{if}\ a \in L\ \textsf{then}\ \bot\ \textsf{else case}\ \mathcal{S}\ a\ \textsf{of}\ \bot \Rightarrow \bot\ |\ \lfloor \mathit{writeable} \rfloor \Rightarrow \lfloor a \in A \vee \mathit{writeable} \rfloor$

The read-modify-write instruction first adds the current value at address a to temporary t and then checks the store condition *cond* on the temporaries. If it fails this read is the final result of the operation. Otherwise the store is performed. The resulting value of the temporary t is specified by the function *ret* which considers both the old and new value as input. As the read-modify-write instruction is an interlocked operation which flushes the store buffer as a side effect the dirty flag $\mathcal{D}$ is reset. The other effects on the ghost state and the safety sideconditions are the same as for the volatile read and volatile write, respectively.

The only effect of the fence instruction in the system without store buffer is to reset the dirty flag.

The ghost instruction GHOST A L R W allows to transfer ownership when no write is involved i.e., when merely reading from memory. It has the same safety requirements as the corresponding parts in the write instructions.

4.3 Reduction

The reduction theorem we aim at reduces a computation of a machine with store buffers to a sequential consistent computation of the virtual machine. We formulate this as a simulation theorem which states that a computation of the store buffer machine $(ts_{\mathsf{sb}}, m) \overset{\mathsf{sb}}{\Rightarrow}^{*} (ts_{\mathsf{sb}}', m')$ can be simulated by a computation of the virtual machine $(ts, m, \mathcal{S}) \overset{\mathsf{v}}{\Rightarrow}^{*} (ts', m', \mathcal{S}')$. The main theorem only considers computations that start in an initial configuration where all store buffers are empty and end in a configuration where all store buffers are empty again. A configuration of the store buffer machine is obtained from a virtual configuration by removing all ghost components and assuming empty store buffers. This coupling relation between the thread configurations is written as $ts_{\mathsf{sb}} \sim ts$. Moreover, the precondition $\mathsf{initial}_{\mathsf{v}}\ ts\ \mathcal{S}$ ensures that the ghost state of the initial configuration of the virtual machine is properly initialized: the ownership sets of the threads are distinct, an address marked as read-only (according to $\mathcal{S}$) is unowned and every unowned address is shared. Finally with $\mathsf{safe\text{-}reach}\ (ts, m, S)$ we ensure conformance to the programming discipline by assuming that all reachable configuration in the virtual machine are safe (according to the rules in Figure 7).

Theorem 1 (Reduction).

$$(ts_{\mathsf{sb}}, m) \overset{\mathsf{sb}}{\Rightarrow}^{*} (ts_{\mathsf{sb}}', m') \wedge \mathsf{empty\text{-}store\text{-}buffers}\ ts_{\mathsf{sb}}' \wedge ts_{\mathsf{sb}} \sim ts\ \wedge \mathsf{initial}_{\mathsf{v}}\ ts\ \mathcal{S} \wedge$$
$$\mathsf{safe\text{-}reach}\ (ts, m, \mathcal{S}) \longrightarrow$$
$$\exists\, ts'\, \mathcal{S}'.\ (ts, m, \mathcal{S}) \overset{\mathsf{v}}{\Rightarrow}^{*} (ts', m', \mathcal{S}') \wedge ts_{\mathsf{sb}}' \sim ts'$$

This theorem captures our intiution that every result that can be obtained from a computation of the store buffer machine can also be obtained by a sequentially consistent computation. However, to prove it we need some generalizations that we sketch in the following sections. First of all the theorem is not inductive as we do not consider arbitrary intermediate configurations but only those where all store buffers are empty. For intermediate confiugrations the coupling relation becomes more involved. The major obstacle is that a volatile read (from memory) can overtake non-volatile writes that are still in the store-buffer and have not yet emerged to memory. Keep in mind that our programming discipline only ensures that no *volatile* writes can be in the store buffer the moment we do a volatile read, outstanding non-volatile writes are allowed. This reordering of operations is reflected in the coupling relation for intermediate configurations as discussed in the following section.

5 Building blocks of the proof

A corner stone of the proof is a proper coupling relation between an *intermediate* configuration of a machine with store buffers and the virtual machine without store buffers. It allows us to simulate every computation step of the store buffer machine by a sequence of steps (potentially empty) on the virtual machine. This transformation is essentially a sequentialization of the trace of the store buffer machine. When a thread of the store buffer machine executes a non-volatile operation, it only accesses memory which is not modified by any other thread (it is either owned or read-only). Although a non-volatile store is buffered, we can immediately execute it on the virtual machine, as there is no competing store of another thread. However, with volatile writes we have to be careful, since concurrent threads may also compete with some volatile write to the same address. At the moment the volatile write enters the store buffer we do not yet know when it will be issued to memory and how it is ordered relatively to other outstanding writes of other threads. We therefore have to suspend the write on the virtual machine from the moment it enters the store buffer to the moment it is issued to memory. For volatile reads our programming discipline guarantees that there is no volatile write in the store buffer by flushing the store buffer if necessary. So there are at most some outstanding non-volatile writes in the store buffer, which are already executed on the virtual machine, as described before. One simple coupling relation one may think of is to suspend the whole store buffer as not yet executed intructions of the virtual machine. However, consider the following scenario. A thread is reading from a volatile address. It can still have non-volatile writes in its store buffer. Hence the read would be suspended in the virutal machine, and other writes to the address (e.g. interlocked or volatile writes of another thread) could invalidate the value. Altogether this suggests the following refined coupling relation: the state of the virtual machine is obtained from the state of the store buffer machine, by executing each store buffer until we reach the first volatile write. The remaining store buffer entries are suspended as instructions. As we only execute non volatile writes the order in which we execute the store buffers should be irrelevant. This coupling relation allows a volatile read to be simulated immediately on the virtual machine as it happens on the store buffer machine.

From the viewpoint of the memory the virtual machine is ahead of the store buffer machine, as leading non-volatile writes already took effect on the memory of the virtual machine while they are still pending in the store buffer. However, if there is a volatile write in the store buffer the corresponding thread in the virtual machine is suspended until the write leaves the store buffer. So from the viewpoint of the already executed instructions the store buffer machine is ahead of the virtual machine. To keep track of this delay we introduce a variant of the store buffer machine below, which maintains the history of executed instructions in the store buffer (including reads and program steps). Moreover, the intermediate machine also maintains the ghost state of the virtual machine to support the coupling relation. We also introduce a refined version of the virutal machine below, which we try to motivate now. Esentially the programming discipline only allows races between volatile (or interlocked) operations. By race we mean two competing memory accesses of different threads of which at least one is a write. For example the discipline guarantees that a volatile read may not be invalidated by a non-volatile write of another thread. While proving the simulation theorem this manifests in the argument that a read of the store-buffer machine and the virtual machine sees the same value in both machines: the value seen by a read in the store buffer machine stays valid as long as it has not yet made its way out in the virtual machine. To rule out certain races from the execution traces we make use of the programming discipline, which is formalized in the safety of all reachable configurations of the virtual machine. Some races can be ruled out by continuing the computation of the virtual machine until we reach a safety violation. However, some cannot be ruled out by the future computation of the current trace, but can be invalidated by a safety violation of another trace that deviated from the current one at some point in the past. Consider two threads. Thread 1 attempts to do a volatile read from address a which is currently owned (and not shared) by thread 2, which attempts to do a non-volatile write on a with value 42 and then release the address. In this configuration there is already a safety violation. Thread 1 is not allowed to perform a volatile read from an address that is not shared. However, when Thread 2 has executed his update and has released ownership (both are non-volatile operations) there is no safety violation anymore. Unfortunately this is the state of the virtual machine when we consider the instructions of Thread 2 to be in the store buffer. The store buffer machine and the virtual machine are out of sync. Whereas in the virtual machine Thread 1 will already read 42 (all non-volatile writes are already executed in the virtual machine), the non-volatile write may still be pending in the store buffer of Thread 2 and hence Thread 1 reads the old value in the store buffer machine. This kind of issues arise when a thread has released ownership in the middle of non-volatile operations of the virtual machine, but the next volatile write of this thread has not yet made its way out of the store buffer. When another thread races for the released address in this situation there is always another scheduling of the virtual machine where the release has not yet taken place and we get a safety violation. To make these safety violations visible until the next volatile write we introduce another ghost component that keeps track of the released addresses. It is augmented when an ghost operation releases an address and is reset as the next volatile write is reached. Moreover, we refine our rules for safety to take these released addresses into account. For example, a write to an released address of another thread is forbidden. We refer to these refined model as *delayed releases* (as no other thread can acquire the address as long as it is still in the set of released addresses) and to our original model as *free flowing releases* (as the effect of a release immediate takes place at the point of the ghost instruction). Note that this only affects ownership transfer due to the GHOST instruction. Ownership transfer together with volatile (or interlocked) writes happen simultaneously in both models.

Note that the refined rules for delayed releases are just an intermediate step in our proof. They do not have to be considered for the final programming discipline. As sketched above we can show in a separate theorem that a safety violation in a trace with respect to delayed releases implies a safety violation of a (potenitally other) trace with respect to free flowing releases. Both notions of safety collaps in all configurations where there are no released addresses, like the initial state. So if all reachable configurations are safe with respect to free flowing releases they are also safe with respect to delayed releases. This allows us to use the stricter policy of delayed releases for the simulation proof. Before continuing with the coupling relation, we introduce the refined intermediate models for delayed releases and store buffers with history information.

5.1 Intermediate models

We begin with the virtual machine with delayed releases, for which the memory transitions $(is, j, m, \mathcal{D}, \mathcal{O}, \mathcal{R}, \mathcal{S}) \xrightarrow{\mathsf{vd}}_{\mathsf{m}} (is', j', m', \mathcal{D}', \mathcal{O}', \mathcal{R}', \mathcal{S}')$ are defined Figure 8. The additional ghost component $\mathcal{R}$ is a mapping from addresses to a Boolean flag. If an address is in the domain of $\mathcal{R}$ it was released. The boolean flag is considered to figure out if the released address was previously shared or not. In case the flag is True it was shared otherwise not. This subtle distinction is necessary to properly handle volatile reads. A volatile read from an address owned by another thread is fine as long as it is marked as shared. The released addresses $\mathcal{R}$ are reset at every volatile write as well as interlocked operations and the fence instruction. They are augmented at the ghost instruction taking the sharing information into account:

$\mathsf{aug}\ (\mathsf{dom}\ \mathcal{S})\ R\ \mathcal{R} =$

$$\overline{(\text{READ } \mathit{volatile}\ a\ t\ \#\ \mathit{is},\ j,\ m,\ \mathit{ghst}) \xrightarrow{\mathsf{vd}}_{\mathsf{m}} (\mathit{is},\ j(t \mapsto m\ a),\ m,\ \mathit{ghst})}$$

$$\overline{(\text{WRITE } \mathsf{False}\ a\ (D,\ f)\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ m,\ \mathit{ghst}) \xrightarrow{\mathsf{vd}}_{\mathsf{m}} (\mathit{is},\ j,\ m(a := f\ j),\ \mathit{ghst})}$$

$$\frac{\mathit{ghst} = (\mathcal{D},\ \mathcal{O},\ \mathcal{R},\ \mathcal{S}) \qquad \mathit{ghst}' = (\mathsf{True},\ \mathcal{O} \cup A - R,\ \lambda x.\ \bot,\ \mathcal{S} \oplus_W R \ominus_A L)}{(\text{WRITE } \mathsf{True}\ a\ (D,\ f)\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ m,\ \mathit{ghst}) \xrightarrow{\mathsf{vd}}_{\mathsf{m}} (\mathit{is},\ j,\ m(a := f\ j),\ \mathit{ghst}')}$$

$$\frac{\neg\ \mathit{cond}\ (j(t \mapsto m\ a)) \qquad \mathit{ghst} = (\mathcal{D},\ \mathcal{O},\ \mathcal{R},\ \mathcal{S}) \qquad \mathit{ghst}' = (\mathsf{False},\ \mathcal{O},\ \lambda x.\ \bot,\ \mathcal{S})}{(\text{RMW } a\ t\ (D,\ f)\ \mathit{cond}\ \mathit{ret}\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ m,\ \mathit{ghst}) \xrightarrow{\mathsf{vd}}_{\mathsf{m}} (\mathit{is},\ j(t \mapsto m\ a),\ m,\ \mathit{ghst}')}$$

$$\frac{\begin{array}{c}\mathit{cond}\ (j(t \mapsto m\ a)) \qquad j' = j(t \mapsto \mathit{ret}\ (m\ a)\ (f\ (j(t \mapsto m\ a)))) \qquad m' = m(a := f\ (j(t \mapsto m\ a))) \\ \mathit{ghst} = (\mathcal{D},\ \mathcal{O},\ \mathcal{R},\ \mathcal{S}) \qquad \mathit{ghst}' = (\mathsf{False},\ \mathcal{O} \cup A - R,\ \lambda x.\ \bot,\ \mathcal{S} \oplus_W R \ominus_A L)\end{array}}{(\text{RMW } a\ t\ (D,\ f)\ \mathit{cond}\ \mathit{ret}\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ m,\ \mathit{ghst}) \xrightarrow{\mathsf{vd}}_{\mathsf{m}} (\mathit{is},\ j',\ m',\ \mathit{ghst}')}$$

$$\overline{(\text{FENCE}\ \#\ \mathit{is},\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{R},\ \mathcal{S}) \xrightarrow{\mathsf{vd}}_{\mathsf{m}} (\mathit{is},\ j,\ m,\ \mathsf{False},\ \mathcal{O},\ \lambda x.\ \bot,\ \mathcal{S})}$$

$$\frac{\mathit{ghst} = (\mathcal{D},\ \mathcal{O},\ \mathcal{R},\ \mathcal{S}) \qquad \mathit{ghst}' = (\mathcal{D},\ \mathcal{O} \cup A - R,\ \mathsf{aug}\ (\mathsf{dom}\ \mathcal{S})\ R\ \mathcal{R},\ \mathcal{S} \oplus_W R \ominus_A L)}{(\text{GHOST } A\ L\ R\ W\ \#\ \mathit{is},\ j,\ m,\ \mathit{ghst}) \xrightarrow{\mathsf{vd}}_{\mathsf{m}} (\mathit{is},\ j,\ m,\ \mathit{ghst}')}$$

Fig. 8: Memory transitions of the virtual machine with delayed releases

$(\lambda a.\ \mathsf{if}\ a \in R\ \mathsf{then\ case}\ \mathcal{R}\ a\ \mathsf{of}\ \bot \Rightarrow \lfloor a \in \mathsf{dom}\ \mathcal{S} \rfloor \mid \lfloor s \rfloor \Rightarrow \lfloor s \wedge a \in \mathsf{dom}\ \mathcal{S} \rfloor$
$\quad \mathsf{else}\ \mathcal{R}\ a)$

If an address is freshly released ($a \in R$ and $\mathcal{R}\ a = \bot$) the flag is set according to $\mathsf{dom}\ \mathcal{S}$. Otherwise the flag becomes $\lfloor\mathsf{False}\rfloor$ in case the released address is currently unshared. Note that with this definition $\mathcal{R}\ a = \lfloor\mathsf{False}\rfloor$ stays stable upon every new release and we do not loose information about a release of an unshared address.

The global transition $(ts,\ m,\ s) \overset{\mathsf{vd}}{\Rightarrow} (ts',\ m',\ s')$ are analogous to the rules in Figure 5 replacing the memory transtions with the refined version for delayed releases.

The safety judgment for delayed releases $\mathcal{O}s,\mathcal{R}s,i \vdash (is,\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd$ is defined in Figure 9. Note the additional component $\mathcal{R}s$ which is the list of release maps of all threads. The rules are strict extensions of the rules in Figure 7: writing or acquiring an address a is only allowed if the address is not in the release set of another thread ($a \notin \mathsf{dom}\ \mathcal{R}s_{[j]}$); reading from an address is only allowed if it is not released by another thread while it was unshared ($\mathcal{R}s_{[j]}\ a \neq \lfloor\mathsf{False}\rfloor$).

For the store buffer machine with history information we not only put writes into the store buffer but also record reads, program steps and ghost operations. This allows us to restore the necessary computation history of the store buffer machine and relate it to the virtual machine which may fall behind the store buffer machine during execution. Altogether an entry in the store buffer is either a

- $\text{READ}_{\mathsf{sb}}$ *volatile* $a\ t\ v$, recording a corresponding read from address a which loaded the value v to temporary t, or a
- $\text{WRITE}_{\mathsf{sb}}$ *volatile* $a\ \mathit{sop}\ v$ for an outstanding write, where operation *sop* evaluated to value v, or of the form
- $\text{PROG}_{\mathsf{sb}}\ p\ p'\ \mathit{is}'$, recording a program transition from p to p' which issued instructions is', or of the form
- $\text{GHOST}_{\mathsf{sb}}\ A\ L\ R\ W$, recording a corresponding ghost operation.

As defined in Figure 10 a write updates the memory when it exits the store buffer, all other store buffer entries may only have an effect on the ghost state. The effect on the ownership

$$\frac{\begin{array}{c} a \in \mathcal{O} \lor a \in \textsf{read-only}\ \mathcal{S} \lor \mathit{volatile} \land a \in \textsf{dom}\ \mathcal{S} \qquad \forall j{<}|\mathcal{O}s|.\ i \neq j \longrightarrow \mathcal{R}s_{[j]}\ a \neq \lfloor\textsf{False}\rfloor \\ \neg\ \mathit{volatile} \longrightarrow (\forall j{<}|\mathcal{O}s|.\ i \neq j \longrightarrow a \notin \textsf{dom}\ \mathcal{R}s_{[j]}) \qquad \mathit{volatile} \longrightarrow \neg\ \mathcal{D} \end{array}}{\mathcal{O}s,\mathcal{R}s,i \vdash (\textsc{Read}\ \mathit{volatile}\ a\ t\ \#\ \mathit{is},\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd}$$

$$\frac{a \in \mathcal{O} \qquad a \notin \textsf{dom}\ \mathcal{S} \qquad \forall j{<}|\mathcal{O}s|.\ i \neq j \longrightarrow a \notin \textsf{dom}\ \mathcal{R}s_{[j]}}{\mathcal{O}s,\mathcal{R}s,i \vdash (\textsc{Write}\ \textsf{False}\ a\ (D,\ f)\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd}$$

$$\frac{\begin{array}{c} \forall j{<}|\mathcal{O}s|.\ i \neq j \longrightarrow a \notin \mathcal{O}s_{[j]} \cup \textsf{dom}\ \mathcal{R}s_{[j]} \\ a \notin \textsf{read-only}\ \mathcal{S} \qquad \forall j{<}|\mathcal{O}s|.\ i \neq j \longrightarrow A \cap (\mathcal{O}s_{[j]} \cup \textsf{dom}\ \mathcal{R}s_{[j]}) = \emptyset \\ A \subseteq \textsf{dom}\ \mathcal{S} \cup \mathcal{O} \qquad L \subseteq A \qquad R \subseteq \mathcal{O} \qquad A \cap R = \emptyset \end{array}}{\mathcal{O}s,\mathcal{R}s,i \vdash (\textsc{Write}\ \textsf{True}\ a\ (D,\ f)\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd}$$

$$\frac{\neg\ \mathit{cond}\ (j(t \mapsto m\ a)) \qquad a \in \textsf{dom}\ \mathcal{S} \cup \mathcal{O} \qquad \forall j{<}|\mathcal{O}s|.\ i \neq j \longrightarrow \mathcal{R}s_{[j]}\ a \neq \lfloor\textsf{False}\rfloor}{\mathcal{O}s,\mathcal{R}s,i \vdash (\text{RMW}\ a\ t\ (D,\ f)\ \mathit{cond}\ \mathit{ret}\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd}$$

$$\frac{\begin{array}{c} \mathit{cond}\ (j(t \mapsto m\ a)) \qquad a \in \textsf{dom}\ \mathcal{S} \cup \mathcal{O} \qquad \forall j{<}|\mathcal{O}s|.\ i \neq j \longrightarrow a \notin \mathcal{O}s_{[j]} \cup \textsf{dom}\ \mathcal{R}s_{[j]} \\ a \notin \textsf{read-only}\ \mathcal{S} \qquad \forall j{<}|\mathcal{O}s|.\ i \neq j \longrightarrow A \cap (\mathcal{O}s_{[j]} \cup \textsf{dom}\ \mathcal{R}s_{[j]}) = \emptyset \\ A \subseteq \textsf{dom}\ \mathcal{S} \cup \mathcal{O} \qquad L \subseteq A \qquad R \subseteq \mathcal{O} \qquad A \cap R = \emptyset \end{array}}{\mathcal{O}s,\mathcal{R}s,i \vdash (\text{RMW}\ a\ t\ (D,\ f)\ \mathit{cond}\ \mathit{ret}\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd}$$

$$\frac{}{\mathcal{O}s,\mathcal{R}s,i \vdash (\textsc{Fence}\ \#\ \mathit{is},\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd}$$

$$\frac{\begin{array}{c} A \subseteq \textsf{dom}\ \mathcal{S} \cup \mathcal{O} \\ L \subseteq A \qquad R \subseteq \mathcal{O} \qquad A \cap R = \emptyset \qquad \forall j{<}|\mathcal{O}s|.\ i \neq j \longrightarrow A \cap (\mathcal{O}s_{[j]} \cup \textsf{dom}\ \mathcal{R}s_{[j]}) = \emptyset \end{array}}{\mathcal{O}s,\mathcal{R}s,i \vdash (\textsc{Ghost}\ A\ L\ R\ W\ \#\ \mathit{is},\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd}$$

$$\mathcal{O}s,\mathcal{R}s,i \vdash ([],\ j,\ m,\ \mathcal{D},\ \mathcal{O},\ \mathcal{S})\ \surd$$

Fig. 9: Safe configurations of a virtual machine (delayed-releases)

$$\frac{}{(m,\ \textsc{Write}_{\textsf{sb}}\ \textsf{False}\ a\ \mathit{sop}\ v\ A\ L\ R\ W\ \#\ \mathit{sb},\ \mathcal{O},\ \mathcal{R},\ \mathcal{S}) \rightarrow_{\textsf{sbh}} (m(a := v),\ \mathit{sb},\ \mathcal{O},\ \mathcal{R},\ \mathcal{S})}$$

$$\frac{\mathcal{O}' = \mathcal{O} \cup A - R \qquad \mathcal{S}' = \mathcal{S} \oplus_W R \ominus_A L}{(m,\ \textsc{Write}_{\textsf{sb}}\ \textsf{True}\ a\ \mathit{sop}\ v\ A\ L\ R\ W\ \#\ \mathit{sb},\ \mathcal{O},\ \mathcal{R},\ \mathcal{S}) \rightarrow_{\textsf{sbh}} (m(a := v),\ \mathit{sb},\ \mathcal{O}',\ \lambda x.\ \bot,\ \mathcal{S}')}$$

$$\frac{}{(m,\ \textsc{Read}_{\textsf{sb}}\ \mathit{volatile}\ a\ t\ v\ \#\ \mathit{sb},\ \mathcal{O},\ \mathcal{R},\ \mathcal{S}) \rightarrow_{\textsf{sbh}} (m,\ \mathit{sb},\ \mathcal{O},\ \mathcal{R},\ \mathcal{S})}$$

$$\frac{}{(m,\ \textsc{Prog}_{\textsf{sb}}\ p\ p'\ \mathit{is}\ \#\ \mathit{sb},\ \mathcal{O},\ \mathcal{R},\ \mathcal{S}) \rightarrow_{\textsf{sbh}} (m,\ \mathit{sb},\ \mathcal{O},\ \mathcal{R},\ \mathcal{S})}$$

$$\frac{\mathcal{O}' = \mathcal{O} \cup A - R \qquad \mathcal{R}' = \textsf{aug}\ (\textsf{dom}\ \mathcal{S})\ R\ \mathcal{R} \qquad \mathcal{S}' = \mathcal{S} \oplus_W R \ominus_A L}{(m,\ \textsc{Ghost}_{\textsf{sb}}\ A\ L\ R\ W\ \#\ \mathit{sb},\ \mathcal{O},\ \mathcal{R},\ \mathcal{S}) \rightarrow_{\textsf{sbh}} (m,\ \mathit{sb},\ \mathcal{O}',\ \mathcal{R}',\ \mathcal{S}')}$$

Fig. 10: Store buffer transitions with history

information is analogous to the corresponding operations in the virtual machine. The memory transitions defined in Figure 11 are straightforward extensions of the store buffer transitions of Figure 11 augmented with ghost state and recording history information in the store buffer. Note how we deal with the ghost state. Only the dirty flag is updated when the instruction enters the store buffer, the ownership transfer takes effect when the instruction leaves the store buffer. The global transitions $(ts_{\mathsf{sbh}},\, m,\, \mathcal{S}) \overset{\mathsf{sbh}}{\Rightarrow} (ts_{\mathsf{sbh}}',\, m',\, \mathcal{S}')$

$$\frac{v = (\mathsf{case\ buffered\text{-}val}\ sb\ a\ \mathsf{of}\ \bot \Rightarrow m\ a \mid \lfloor v' \rfloor \Rightarrow v') \qquad sb' = sb\ @\ [\textsc{Read}_{\mathsf{sb}}\ volatile\ a\ t\ v]}{(\textsc{Read}\ volatile\ a\ t\ \#\ is,\, j,\, sb,\, m,\, ghst) \xrightarrow{\mathsf{sbh}}_{\mathsf{m}} (is,\, j(t \mapsto v),\, sb',\, m,\, ghst)}$$

$$\frac{sb' = sb\ @\ [\textsc{Write}_{\mathsf{sb}}\ \mathsf{False}\ a\ (D,\, f)\ (f\ j)\ A\ L\ R\ W]}{(\textsc{Write}\ \mathsf{False}\ a\ (D,\, f)\ A\ L\ R\ W\ \#\ is,\, j,\, sb,\, m,\, ghst) \xrightarrow{\mathsf{sbh}}_{\mathsf{m}} (is,\, j,\, sb',\, m,\, ghst)}$$

$$\frac{\begin{array}{c} sb' = sb\ @\ [\textsc{Write}_{\mathsf{sb}}\ \mathsf{True}\ a\ (D,\, f)\ (f\ j)\ A\ L\ R\ W] \\ ghst = (\mathcal{D},\, \mathcal{O},\, \mathcal{R},\, \mathcal{S}) \qquad ghst' = (\mathsf{True},\, \mathcal{O},\, \mathcal{R},\, \mathcal{S}) \end{array}}{(\textsc{Write}\ \mathsf{True}\ a\ (D,\, f)\ A\ L\ R\ W\ \#\ is,\, j,\, sb,\, m,\, ghst) \xrightarrow{\mathsf{sbh}}_{\mathsf{m}} (is,\, j,\, sb',\, m,\, ghst')}$$

$$\frac{\neg\ cond\ (j(t \mapsto m\ a)) \qquad ghst = (\mathcal{D},\, \mathcal{O},\, \mathcal{R},\, \mathcal{S}) \qquad ghst' = (\mathsf{False},\, \mathcal{O},\, \lambda x.\ \bot,\, \mathcal{S})}{(\mathrm{RMW}\ a\ t\ (D,\, f)\ cond\ ret\ A\ L\ R\ W\ \#\ is,\, j,\, [],\, m,\, ghst) \xrightarrow{\mathsf{sbh}}_{\mathsf{m}} (is,\, j(t \mapsto m\ a),\, [],\, m,\, ghst')}$$

$$\frac{\begin{array}{c} cond\ (j(t \mapsto m\ a)) \qquad j' = j(t \mapsto ret\ (m\ a)\ (f\ (j(t \mapsto m\ a)))) \qquad m' = m(a := f\ (j(t \mapsto m\ a))) \\ ghst = (\mathcal{D},\, \mathcal{O},\, \mathcal{R},\, \mathcal{S}) \qquad ghst' = (\mathsf{False},\, \mathcal{O} \cup A - R,\, \lambda x.\ \bot,\, \mathcal{S} \oplus_{\mathrm{W}} R \ominus_{\mathrm{A}} L) \end{array}}{(\mathrm{RMW}\ a\ t\ (D,\, f)\ cond\ ret\ A\ L\ R\ W\ \#\ is,\, j,\, [],\, m,\, ghst) \xrightarrow{\mathsf{sbh}}_{\mathsf{m}} (is,\, j',\, [],\, m',\, ghst')}$$

$$\frac{}{(\textsc{Fence}\ \#\ is,\, j,\, [],\, m,\, \mathcal{D},\, \mathcal{O},\, \mathcal{R},\, \mathcal{S}) \xrightarrow{\mathsf{sbh}}_{\mathsf{m}} (is,\, j,\, [],\, m,\, \mathsf{False},\, \mathcal{O},\, \lambda x.\ \bot,\, \mathcal{S})}$$

$$\frac{}{(\textsc{Ghost}\ A\ L\ R\ W\ \#\ is,\, j,\, sb,\, m,\, G) \xrightarrow{\mathsf{sbh}}_{\mathsf{m}} (is,\, j,\, sb\ @\ [\textsc{Ghost}_{\mathsf{sb}}\ A\ L\ R\ W],\, m,\, G)}$$

Fig. 11: Memory transitions of store buffer machine with history

are analogous to the rules in Figure 2 replacing the memory transtions and store buffer transtiontions accordingly.

5.2 Coupling relation

After this introduction of the immediate models we can proceed to the details of the coupling relation, which relates configurations of the store buffer machine with histroy and the virtual machine with delayed releases. Remember the basic idea of the coupling relation: the state of the virtual machine is obtained from the state of the store buffer machine, by executing each store buffer until we reach the first volatile write. The remaining store buffer entries are suspended as instructions. The instructions now also include the history entries for reads, program steps and ghost operations. The suspended reads are not yet visible in the temporaries of the virtual machine. Similar the ownership effects (and program steps) of the suspended operations are not yet visible in the virtual machine. The coupling relation between the store buffer machine and the virtual machine is illustrated in Figure 12. The threads issue instructions to the store buffers from the right and the instructions emerge from the store buffers to main memory from the left. The dotted line illustrates the state of the virtual machines memory. It is obtained from the memory of the store buffer machine by executing the purely non-volatile prefixes of the store buffers. The remaining entries of the store buffer are still (suspended) instructions in the virtual machine.

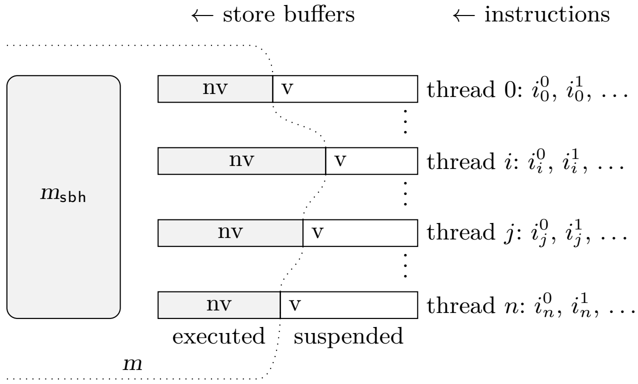

Fig. 12: Illustration of coupling relation

Consider the following configuration of a thread $ts_{\mathsf{sbh}[j]}$ in the store buffer machine, where i_{k} are the instructions and s_{k} the store buffer entries. Let s_{v} be the first volatile write in the store buffer. Keep in mind that new store buffer entries are appended to the end of the list and entries exit the store buffer and are issued to memory from the front of the list.

$$ts_{\mathsf{sbh}[j]} = (p, [i_1, \ldots, i_{\mathsf{n}}], j, [s_1, \ldots, s_{\mathsf{v}}, s_{\mathsf{v}+1}, \ldots, s_{\mathsf{m}}], \mathcal{D}, \mathcal{O}, \mathcal{R})$$

The corresponding configuration $ts_{[j]}$ in the virtual machine is obtained by suspending all store buffer entries beginning at s_{v} to the front of the instructions. A store buffer $\textsc{Read}_{\mathsf{sb}}$ / $\textsc{Write}_{\mathsf{sb}}$ / $\textsc{Ghost}_{\mathsf{sb}}$ is converted to a $\textsc{Read}$ / $\textsc{Write}$ / $\textsc{Ghost}$ instruction. We take the freedom to make this coercion implicit in the example. The store buffer entries preceding s_{v} have already made their way to memory, whereas the suspended read operations are not yet visible in the temporaries j'. Similar, the suspended updates to the ownership sets and dirty flag are not yet recorded in $\mathcal{O}'$, $\mathcal{R}'$ and $\mathcal{D}'$.

$$ts_{[j]} = (p, [s_{\mathsf{v}}, s_{\mathsf{v}+1}, \ldots, s_{\mathsf{m}}, i_1, \ldots, i_{\mathsf{n}}], j', \mathcal{D}', \mathcal{O}', \mathcal{R}')$$

This example illustrates that the virtual machine falls behind the store buffer machine in our simulation, as store buffer instructions are suspended and reads (and ghost operations) are delayed and not yet visible in the temporaries (and the ghost state). This delay can also propagate to the level of the programming language, which communicates with the memory system by reading the temporaries and issuing new instructions. For example the control flow can depend on the temporaries, which store the result of branching conditions. It may happen that the store buffer machine already has evaluated the branching condition by referring to the values in the store buffer, whereas the virtual machine still has to wait. Formally this manifests in still undefined temporaries. Now consider that the program in the store buffer machine makes a step from p to (p', is'), which results in a thread configuration where the program state has switched to p', the instructions is' are appended and the program step is recorded in the store buffer:

$$ts_{\mathsf{sbh}}{'}_{[j]} = (p', [i_1, \ldots, i_{\mathsf{n}}] @ is', j, [s_1, \ldots, s_{\mathsf{v}}, \ldots, s_{\mathsf{m}}, \textsc{Prog}_{\mathsf{sb}}\ p\ p'\ is'], \mathcal{D}, \mathcal{O}, \mathcal{R})$$

The virtual machine however makes no step, since it still has to evaluate the suspended instructions before making the program step. The instructions is' are not yet issued and the program state is still p. We also take these program steps into account in our final coupling relation $(ts_{\mathsf{sbh}}, m_{\mathsf{sbh}}, \mathcal{S}_{\mathsf{sbh}}) \sim (ts, m, \mathcal{S})$, defined in Figure 13. We denote the already simulated store buffer entries by *execs* and the suspended ones by *suspends*. The function instrs converts them back to instructions, which are a prefix of the instructions of the virtual

$$
\frac{\begin{array}{l}
m = \text{exec-all-until-volatile-write}\ ts_{\mathsf{sbh}}\ m_{\mathsf{sbh}} \\
\mathcal{S} = \text{share-all-until-volatile-write}\ ts_{\mathsf{sbh}}\ \mathcal{S}_{\mathsf{sbh}} \qquad |ts_{\mathsf{sbh}}| = |ts| \\
\forall\, i{<}|ts_{\mathsf{sbh}}|. \\
\quad \mathtt{let}\ (p_{\mathsf{sbh}},\ is_{\mathsf{sbh}},\ j_{\mathsf{sbh}},\ sb,\ \mathcal{D}_{\mathsf{sbh}},\ \mathcal{O}_{\mathsf{sbh}},\ \mathcal{R}_{\mathsf{sbh}}) = ts_{\mathsf{sbh}[i]}; \\
\qquad execs = \text{takeWhile not-volatile-write}\ sb; \\
\qquad suspends = \text{dropWhile not-volatile-write}\ sb \\
\quad \mathtt{in}\ \exists\, is\ \mathcal{D}.\ \text{instrs}\ suspends\ @\ is_{\mathsf{sbh}} = is\ @\ \text{prog-instrs}\ suspends\ \wedge \\
\qquad\qquad \mathcal{D}_{\mathsf{sbh}} = (\mathcal{D} \vee \text{refs volatile-Write}\ sb \neq \emptyset)\ \wedge \\
\qquad\qquad ts_{[i]} = \\
\qquad\qquad (\text{hd-prog}\ p_{\mathsf{sbh}}\ suspends,\ is,\ j_{\mathsf{sbh}}\restriction_{(-\ \text{read-tmps}\ suspends)},\ \mathcal{D}, \\
\qquad\qquad\ \text{acquire}\ execs\ \mathcal{O}_{\mathsf{sbh}},\ \text{release}\ execs\ (\text{dom}\ \mathcal{S}_{\mathsf{sbh}})\ \mathcal{R}_{\mathsf{sbh}})
\end{array}}{(ts_{\mathsf{sbh}},\ m_{\mathsf{sbh}},\ \mathcal{S}_{\mathsf{sbh}}) \sim (ts,\ m,\ \mathcal{S})}
$$

Fig. 13: Coupling relation

machine. We collect the additional instructions which were issued by program instructions but still recorded in the remainder of the store buffer with function prog-instrs. These instructions have already made their way to the instructions of the store buffer machine but not yet on the virtual machine. This situation is formalized as instrs *suspends* @ is_{sbh} = *is* @ prog-instrs *suspends*, where *is* are the instructions of the virtual machine. The program state of the virtual machine is either the same as in the store buffer machine or the first program state recorded in the suspended part of the store buffer. This state is selected by hd-prog. The temporaries of the virtual machine are obtained by removing the suspended reads from *j*. The memory is obtained by executing all store buffers until the first volatile write is hit, excluding it. Thereby only non-volatile writes are executed, which are all thread local, and hence could be executed in any order with the same result on the memory. Function exec-all-until-volatile-write executes them in order of appearance. Similarly the sharing map of the virtual machine is obtained by executing all store buffers until the first volatile write via the function share-all-until-volatile-write. For the local ownership set $\mathcal{O}_{\mathsf{sbh}}$ the auxiliary function acquire calculates the outstanding effect of the already simulated parts of the store buffer. Analogously release calculates the effect for the released addresses $\mathcal{R}_{\mathsf{sbh}}$.

5.3 Simulation

Theorem 2 is our core inductive simulation theorem. Provided that all reachable states of the virtual machine (with delayed releases) are safe, a step of the store buffer machine (with history) can be simulated by a (potentially empty) sequence of steps on the virtual machine, maintaining the coupling relation and an invariant on the configurations of the store buffer machine.

Theorem 2 (Simulation).

$$
\begin{array}{l}
(ts_{\mathsf{sbh}},\ m_{\mathsf{sbh}},\ \mathcal{S}_{\mathsf{sbh}}) \overset{\mathsf{sbh}}{\Rightarrow} (ts_{\mathsf{sbh}}',\ m_{\mathsf{sbh}}',\ \mathcal{S}_{\mathsf{sbh}}') \wedge (ts_{\mathsf{sbh}},\ m_{\mathsf{sbh}},\ \mathcal{S}_{\mathsf{sbh}}) \sim (ts,\ m,\ \mathcal{S}) \wedge \\
\quad \text{safe-reach-delayed}\ (ts,\ m,\ \mathcal{S}) \wedge \text{invariant}\ ts_{\mathsf{sbh}}\ \mathcal{S}_{\mathsf{sbh}}\ m_{\mathsf{sbh}} \longrightarrow \\
\quad\ \text{invariant}\ ts_{\mathsf{sbh}}'\ \mathcal{S}_{\mathsf{sbh}}'\ m_{\mathsf{sbh}}' \wedge \\
\quad (\exists\, ts'\ \mathcal{S}'\ m'.\ (ts,\ m,\ \mathcal{S}) \overset{\mathsf{vd}}{\Rightarrow}^{*} (ts',\ m',\ \mathcal{S}') \wedge (ts_{\mathsf{sbh}}',\ m_{\mathsf{sbh}}',\ \mathcal{S}_{\mathsf{sbh}}') \sim (ts',\ m',\ \mathcal{S}'))
\end{array}
$$

In the following we discuss the invariant invariant $ts_{\mathsf{sbh}}\ \mathcal{S}_{\mathsf{sbh}}\ m_{\mathsf{sbh}}$, where we commonly refer to a thread configuration $ts_{\mathsf{sbh}[i]} = (p,\ is,\ j,\ sb,\ \mathcal{D},\ \mathcal{O},\ \mathcal{R})$ for $i < |ts_{\mathsf{sbh}}|$. By outstanding references we refer to read and write operations in the store buffer. The invariant is a conjunction of several sub-invariants grouped by their content:

$$
\text{invariant}\ ts_{\mathsf{sbh}}\ \mathcal{S}_{\mathsf{sbh}}\ m_{\mathsf{sbh}} \equiv \text{ownership-inv}\ \mathcal{S}_{\mathsf{sbh}}\ ts_{\mathsf{sbh}} \wedge \text{sharing-inv}\ \mathcal{S}_{\mathsf{sbh}}\ ts_{\mathsf{sbh}} \wedge
$$

temporaries-inv ts_{sbh} ∧ data-dependency-inv ts_{sbh} ∧ history-inv ts_{sbh} m_{sbh} ∧ flush-inv ts_{sbh} ∧ *valid* ts_{sbh}

Ownership. (i) For every thread all outstanding non-volatile references have to be owned or refer to read-only memory. (ii) Every outstanding volatile write is not owned by any other thread. (iii) Outstanding accesses to read-only memory are not owned. (iv) The ownership sets of every two different threads are distinct.

Sharing. (i) All outstanding non volatile writes are unshared. (ii) All unowned addresses are shared. (iii) No thread owns read-only memory. (iv) The ownership annotations of outstanding ghost and write operations are consistent (e.g., released addresses are owned at the point of release). (v) There is no outstanding write to read-only memory.

Temporaries. Temporaries are modeled as an unlimited store for temporary registers. We require certain distinctness and freshness properties for each thread. (i) The temporaries referred to by read instructions are distinct. (ii) The temporaries referred to by reads in the store buffer are distinct. (iii) Read and write temporaries are distinct. (iv) Read temporaries are fresh, i.e., are not in the domain of *j*.

Data dependency. Data dependency means that store operations may only depend on *previous* read operations. For every thread we have: (i) Every operation (D, f) in a write instruction or a store buffer write is valid according to valid-sop (D, f), i.e., function f only depends on domain D. (ii) For every suffix of the instructions of the form WRITE *volatile a* (D, f) A L R W # *is* the domain D is distinct from the temporaries referred to by future read instructions in *is*. (iii) The outstanding writes in the store buffer do not depend on the read temporaries still in the instruction list.

History. The history information of program steps and read operations we record in the store buffer have to be consistent with the trace. For every thread: (i) The value stored for a non volatile read is the same as the last write to the same address in the store buffer or the value in memory, in case there is no write in the buffer. (ii) All reads have to be clean. This results from our flushing policy. Note that the value recorded for a volatile read in the initial part of the store buffer (before the first volatile write), may become stale with respect to the memory. Remember that those parts of the store buffer are already executed in the virtual machine and thus cause no trouble. (iii) For every read the recorded value coincides with the corresponding value in the temporaries. (iv) For every $\text{WRITE}_{\mathsf{sb}}$ *volatile a* (D, f) v A L R W the recorded value v coincides with f j, and domain D is subset of dom j and is distinct from the following read temporaries. Note that the consistency of the ownership annotations is already covered by the aforementioned invariants. (v) For every suffix in the store buffer of the form $\text{PROG}_{\mathsf{sb}}$ p_1 p_2 *is*′ # sb', either $p_1 = p$ in case there is no preceding program node in the buffer or it corresponds to the last program state recorded there. Moreover, the program transition $j{\restriction}_{(- \text{ read-tmps } sb')} \vdash p_1 \rightarrow_{\mathsf{p}} (p_2, is')$ is possible, i.e., it was possible to execute the program transition at that point. (vi) The program configuration p coincides with the last program configuration recorded in the store buffer. (vii) As the instructions from a program step are at the one hand appended to the instruction list and on the other hand recorded in the store buffer, we have for every suffix sb' of the store buffer: $\exists$ *is*′. instrs sb' @ *is* = *is*′ @ prog-instrs sb', i.e., the remaining instructions *is* correspond to a suffix of the recorded instructions prog-instrs sb'.

Flushes. If the dirty flag is unset there are no outstanding volatile writes in the store buffer.

Program step. The program-transitions are still a parameter of our model. In order to make the proof work, we have to assume some of the invariants also for the program steps. We allow the program-transitions to employ further invariants on the configurations, these are modeled by the parameter *valid*. For example, in the instantiation later on the program keeps a counter for the temporaries, for each thread. We maintain distinctness of temporaries by restricting all temporaries occurring in the memory system to be below that counter, which is expressed by instantiating *valid*. Program steps, memory steps and store buffer steps have to maintain *valid*. Furthermore we assume the following properties of a program step: (i) The program step generates fresh, distinct read temporaries, that are neither in j nor in the store buffer temporaries of the memory system. (ii) The generated memory instructions respect data dependencies, and are valid according to $\mathsf{valid\text{-}sop}$.

Proof sketch. We do not go into details but rather first sketch the main arguments for simulation of a step in the store buffer machine by a potentially empty sequence of steps in the virtual machine, maintaining the coupling relation. Second we exemplarically focus on some cases to illustrate common arguments in the proof. The first case distinction in the proof is on the global transitions in Figure 2. (i) *Program step*: we make a case distinction whether there is an outstanding volatile write in the store buffer or not. If not the configuration of the virtual machine corresponds to the executed store buffer and we can make the same step. Otherwise the virtual machine makes no step as we have to wait until all volatile writes have exited the store buffer. (ii) *Memory step*: we do case distinction on the rules in Figure 11. For read, non volatile write and ghost instructions we do the same case distinction as for the program step. If there is no outstanding volatile write in the store buffer we can make the step, otherwise we have to wait. When a volatile write enters the store buffer it is suspended until it exists the store buffer. Hence we do no step in the virtual machine. The read-modify-write and the fence instruction can all be simulated immediately since the store buffer has to be empty. (iii) *Store Buffer step*: we do case distinction on the rules in Figure 10. When a read, a non volatile write, a ghost operation or a program history node exits the store buffer, the virtual machine does not have to do any step since these steps are already visible. When a volatile write exits the store buffer, we execute all the suspended operations (including reads, ghost operations and program steps) until the next suspended volatile write is hit. This is possible since all writes are non volatile and thus memory modifications are thread local.

In the following we exemplarically describe some cases in more detail to give an impression on the typical arguments in the proof. We start with a configuration $c_{\mathsf{sbh}} = (ts_{\mathsf{sbh}}, m_{\mathsf{sbh}}, \mathcal{S}_{\mathsf{sbh}})$ of the store buffer machine, where the next instruction to be executed is a read of thread i: $\textsc{Read}_{\mathsf{sb}}$ *volatile* a t. The configuration of the virtual machine is $cfg = (ts, m, \mathcal{S})$. We have to simulate this step on the virtual machine and can make use of the coupling relations $(ts_{\mathsf{sbh}}, m_{\mathsf{sbh}}, \mathcal{S}_{\mathsf{sbh}}) \sim (ts, m, \mathcal{S})$, the invariants $\mathsf{invariant}\ ts_{\mathsf{sbh}}\ \mathcal{S}_{\mathsf{sbh}}\ m_{\mathsf{sbh}}$ and the safety of all reachable states of the virtual machine: $\mathsf{safe\text{-}reach\text{-}delayed}\ (ts, m, \mathcal{S})$. The state of the store buffer machine and the coupling with the volatile machine is depicted in Figure 14. Note that if there are some suspended instructions in thread i, we cannot directly exploit the 'safety of the read', as the virtual machine has not yet reached the state where thread i is poised to do the read. But fortunately we have safety of the virtual machien of all reachable states. Hence we can just execute all suspended instructions of thread i until we reach the read. We refer to this configuration of the virtual machine as $cfg'' = (ts'', m'', \mathcal{S}'')$, which is depicted in Figure 15.

For now we want to consider the case where the read goes to memory and is not forwarded from the store buffer. The value read is $v = m_{\mathsf{sbh}}\ a$. Moreover, we make a case distinction wheter there is an outstanding volatile write in the store buffer of thread i or

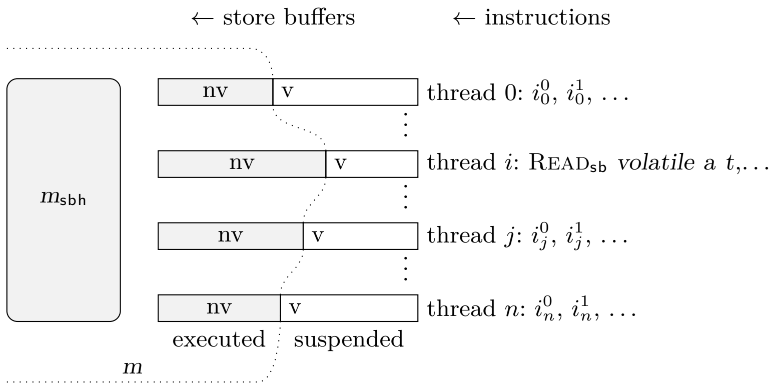

Fig. 14: Thread i poised to read

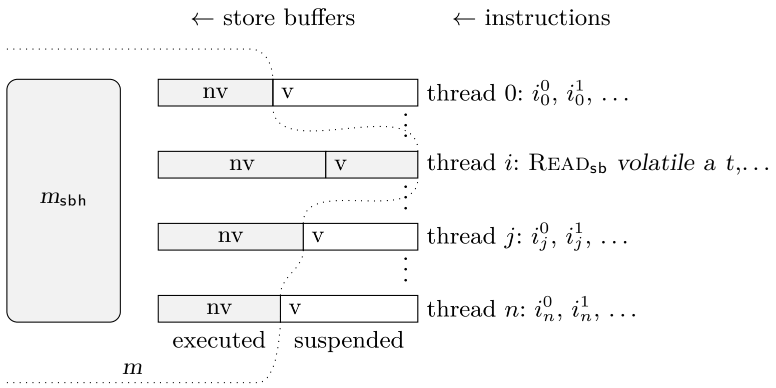

Fig. 15: Forwarded computation of virtual machine

not. This determines if there are suspended instructions in the virtual machine or not. We start with the case where there is no such write. This means that there are no suspended instructions in thread i and therefore $cfg'' = cfg$. We have to show that the virtual machine reads the same value from memory: $v = m\ a$. So what can go wrong? When can the the memory of the virtual machine hold a different value? The memory of the virtual machine is obtained from the memory of the store buffer machine by executing all store buffers until we hit the first volatile write. So if there is a discrepancy in the value this has to come from a non-volatile write in the executed parts of another thread, let us say thread j. This write is marked as x in Figure 16.

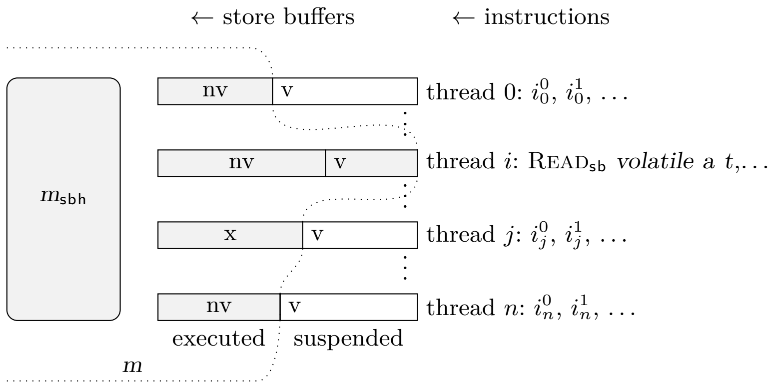

Fig. 16: Conflicting write in thread j (marked x)

We refer to x both for the write operation itself and to characterize the point in time in the computation of the virtual machine where the write was executed. At the point x the write was safe according to rules in Figure 9 for non-volatile writes. So it was owned by thread j and unshared. This knowledge about the safety of write x is preserved in the invariants, namely (Ownership.i) and (Sharing.i). Additionally from invariant (Sharing.v) we know that address a was not read-only at point x. Now we combine this information with the safety of the read of thread i in the current configuration cfg: address a either has to be owned by thread i, or has to be read-only or the read is volatile and a is shared. Additionally there are the constraints on the released addresses which we will exploit below. Let us address all cases step by step. First, we consider that address a is currently owned by thread i. As it was owned by thread j at time x there has to be an release of a in the executed prefix of the store buffer of thread j. This release is recorded in the release set, so we know $a \in \mathsf{dom}\ \mathcal{R}s_{[j]}$. This contradicts the safety of the read. Second, we consider that address a is currently read-only. At time x address a was owned by thread j, unshared and not read-only. Hence there was a release of address a in the executed prefix of the store buffer of j, where it made a transition unshared and owned to shared. With the monotonicity of the release sets this means $a \in \mathsf{dom}\ \mathcal{R}s_{[j]}$, even more precisely $\mathcal{R}s_{[j]}\ a = \lfloor\mathsf{False}\rfloor$. Hence there is no chance to get the read safe (neiter a volatile nor a non-volatile). Third, consider a volatile read and that address a is currently shared. This is ruled out by the same line of reasoning as in the previous case. So ultimately we have ruled out all races that could destroy the value at address a and have shown that we can simulate the step on the virtual machine. This completes the simulation of the case where there is no store buffer forwarding and no volatile write in the store buffer of thread i. The other cases are handled similar. The main arguments are obtained by arguing about safety of configuration cfg'' and exploiting the invariants to rule out conflicting operations in other store buffers. When there is a volatile write in he store buffer of thread i there are still pending suspended instructions in the virtual machine. Hence the virtual machine makes no step and we have to argue that the simulation relation as well as all invariants still hold.

Up to now we have focused on how to simulate the read and in particular on how to argue that the value read in the store buffer machine is the same as the value read in the virtual machine. Besided these simulation properties another major part of the proof is to show that all invariants are maintained. For example if the non-volatile read enters the store buffer we have to argue that this new entry is either owned or refers to an read-only address (Ownership.i). As for the simulation above this follows from safety of the virtual machine in configuration cfg''. However, consider an ghost operation that acquires an address a. From safety of the configuration cfg'' we can only infer that there is no conflicting acquire in the non-volaitle prefixes of the other store buffers. In case an conflicting acquire is in the suspended part of a store buffer of thread j safety of configuration cfg'' is not enough. But as we have safety of all reachable states we can forward the computation of thread j until the conflicting acquire is about to be executed and construct an unsafe state which rules out the conflict.

Last we want to comment on the case where the store buffer takes a step. The major case destinction is wheter a volatile write leaves the store buffer or not. In the former case the virtual machine has to simulate a whole bunch of instructions at once to simulate the store buffer machine up to the next volatile write in the store buffer. In the latter case the virtual machine does no step at all, since the instruction leaving the store buffer is already simulated. In both cases one key argument is commutativity of non-volatile operations with respect to global effects on the memory or the sharing map. Consider a non-volatile store buffer step of thread i. In the configuration of the virtual machine before the store buffer step of thread i, the simulation relation applies the update to the memory and the sharing map of the store buffer machine, within the operations $\mathsf{exec\text{-}all\text{-}until\text{-}volatile\text{-}write}$ and $\mathsf{share\text{-}all\text{-}until\text{-}volatile\text{-}write}$ 'somewhere in the middle' to obtain the memory and the sharing map of the virtual machine. After the store buffer step however, when the non-volatile operations has left the store buffer, the effect is applied to the memory and the sharing map right in the beginning. The invariants and safety sideconditions for non-volatile operations guarantee 'locality' of the operation which manifests in commutativity properties. For example, a non-volatile write is thread local. There is no conflicting write in any other store buffer and hence the write can be safely moved to the beginning.

This conludes the discussion on the proof of Theorem 2. $\square$

The simulation theorem for a single step is inductive and can therefor be extended to arbitrary long computations. Moreover, the coupling relation as well as the invariants become trivial for a initial configuration where all store buffers are empty and the ghost state is setup appropriately. To arrive at our final Theorem 1 we need the following steps:

1. simulate the computation of the store buffer machine $(ts_{\mathsf{sb}}, m) \overset{\mathsf{sb}}{\Rightarrow}^* (ts_{\mathsf{sb}}', m')$ by a computation of a store buffer machine with history $(ts_{\mathsf{sbh}}, m, \mathcal{S}) \overset{\mathsf{sbh}}{\Rightarrow}^* (ts_{\mathsf{sbh}}', m', \mathcal{S}')$,
2. simulate the computation of the store buffer machine with history by a computation of the virtual machine with delayed releases $(ts, m, \mathcal{S}) \overset{\mathsf{v_d}}{\Rightarrow}^* (ts', m', \mathcal{S}')$ by Theorem 2 (extended to the reflexive transitive closure),
3. simulate the computation of the virtual machine with delayed releases by a computation of the virtual machine with free flowing releases $(ts, m, \mathcal{S}) \overset{\mathsf{v}}{\Rightarrow}^* (ts', m', \mathcal{S}')$[5].

[5] Here we are sloppy with ts; strictly we would have to distinguish the thread configurations without the $\mathcal{R}$ component form the ones with the $\mathcal{R}$ component used for delayed releases

Step 1 is trivial since the bookkeeping within the additional ghost and history state does not affect the control flow of the transition systems and can be easily removed. Similar the additional $\mathcal{R}$ ghost component can be ignored in Step 3. However, to apply Theorem 2 in Step 2 we have to convert from safe-reach $(ts, m, \mathcal{S})$ provided by the preconditions of Theorem 1 to the required safe-reach-delayed $(ts, m, \mathcal{S})$. This argument is more involved and we only give a short sketch here. The other direction is trivial as every single case for delayed releases (cf. Figure 9) immediately implies the corresponding case for free flowing releases (cf. Figure 7).

First keep in mind that the predicates ensure that *all* reachable configurations starting from $(ts, m, \mathcal{S})$ are safe, according to the rules for free flowing releases or delayed releases respectively. We show the theorem by contraposition and start with a computation which reaches a configuration c that is unsafe according to the rules for delayed releases and want to show that there has to be a (potentially other) computation (starting from the same initial state) that leads to an unsafe configuration c' accroding to free flowing releases. If c is already unsafe according to free flowing releases we have $c' = c$ and are finished. Otherwise we have to find another unsafe configuration. Via induction on the length of the global computation we can also assume that for all shorter computations both safety notions coincide. A configuration can only be unsafe with respect to delayed releases and safe with respect to free flowing releases if there is a race between two distinct Threads i and j on an address a that is in the release set $\mathcal{R}$ of one of the threads, lets say Thread i. For example Thread j attempts to write to an address a which is in the release set of Thread i. If the release map would be empty there cannot be such an race (it would simulataneously be unsafe with respect to free flowing releases). Now we aim to find a configuration c' that is also reachable from the initial configuration and is unsafe with respect to free flowing releases. Intuitively this is a configuration where Thread i is rewinded to the state just before the release of address a and Thread j is in the same state as in configuration c. Before the release of a the address has to be owned by Thread i, which is unsafe according to free flowing releases as well as delayed releases. So we can argue that either Thread j can reach the same state although Thread i is rewinded or we even hit an unsafe configuration before. What kind of steps can Thread i perform between between the free flowing release point (point of the ghost instruction) and the delayed release point (point of next volatile write, interlocked operation or fence at which the release map is emptied)? How can these actions affect Thread j? Note that the delayed release point is not yet reached as this would empty the release map (which we know not to be empty). Thus Thread i does only perform reads, ghost instructions, program steps or non-volatile writes. All of these instructions of Thread i either have no influence on the computation of Thread j at all (e.g. a read, program step, non-volatile write or irrelevant ghost operation) or may cause a safety violation already in a shorter computation (e.g. acquiring an address that another thread holds). This is fine for our inductive argument. So either we can replay every step of Thread j and reach the final configuration c' which is now also unsafe according to free flowing releases, or we hit a configuration c'' in a shorter computation which violates the rules of delayed as well as free flowing releases (using the induction hypothesis).

6 PIMP

PIMP is a parallel version of IMP [11], a canonical WHILE-language.

An expression e is either (i) Const v, a constant value, (ii) Mem *volatile* a, a (volatile) memory lookup at address a, (iii) Tmp *sop*, reading from the temporaries with a operation *sop* which is an intermediate expression occurring in the transition rules for statements,

(iv) $\textsc{Unop}\ f\ e$, a unary operation where f is a unary function on values, and finally (v) $\textsc{Binop}\ f\ e_1\ e_2$, a binary operation where f is a binary function on values.

A statement s is either (i) $\textsc{Skip}$, the empty statement, (ii) $\textsc{Assign}\ \mathit{volatile}\ a\ e\ A\ L\ R\ W$, a (volatile) assignment of expression e to address expression a, (iii) CAS $a\ c_\mathsf{e}\ s_\mathsf{e}\ A\ L\ R\ W$, atomic compare and swap at address expression a with compare expression c_e and swap expression s_e, (iv) $\textsc{Seq}\ s_1\ s_2$, sequential composition, (v) $\textsc{Cond}\ e\ s_1\ s_2$, the if-then-else statement, (vi) $\textsc{While}\ e\ s$, the loop statement with condition e, (vii) $\mathrm{S}\textsc{Ghost}$, and $\mathrm{S}\textsc{Fence}$ as stubs for the corresponding memory instructions.

The key idea of the semantics is the following: expressions are evaluated by issuing instructions to the memory system, then the program waits until the memory system has made all necessary results available in the temporaries, which allows the program to make another step. Figure 17 defines expression evaluation. The function $\mathsf{used\text{-}tmps}\ e$ calculates

$$
\begin{array}{lcl}
\mathsf{issue\text{-}expr}\ t\ (\textsc{Const}\ v) & = & [] \\
\mathsf{issue\text{-}expr}\ t\ (\textsc{Mem}\ \mathit{volatile}\ a) & = & [\textsc{Read}\ \mathit{volatile}\ a\ t] \\
\mathsf{issue\text{-}expr}\ t\ (\textsc{Tmp}\ (D,\ f)) & = & [] \\
\mathsf{issue\text{-}expr}\ t\ (\textsc{Unop}\ f\ e) & = & \mathsf{issue\text{-}expr}\ t\ e \\
\mathsf{issue\text{-}expr}\ t\ (\textsc{Binop}\ f\ e_1\ e_2) & = & \mathsf{issue\text{-}expr}\ t\ e_1\ @\ \mathsf{issue\text{-}expr}\ (t + \mathsf{used\text{-}tmps}\ e_1)\ e_2
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{eval\text{-}expr}\ t\ (\textsc{Const}\ v) & = & (\emptyset,\ \lambda j.\ v) \\
\mathsf{eval\text{-}expr}\ t\ (\textsc{Mem}\ \mathit{volatile}\ a) & = & (\{t\},\ \lambda j.\ \mathsf{the}\ (j\ t)) \\
\mathsf{eval\text{-}expr}\ t\ (\textsc{Tmp}\ (D,\ f)) & = & (D,\ f) \\
\mathsf{eval\text{-}expr}\ t\ (\textsc{Unop}\ f\ e) & = & \mathtt{let}\ (D,\ f_\mathsf{e}) = \mathsf{eval\text{-}expr}\ t\ e\ \mathtt{in}\ (D,\ \lambda j.\ f\ (f_\mathsf{e}\ j)) \\
\mathsf{eval\text{-}expr}\ t\ (\textsc{Binop}\ f\ e_1\ e_2) & = & \mathtt{let}\ (D_1,\ f_1) = \mathsf{eval\text{-}expr}\ t\ e_1; \\
 & & \qquad (D_2,\ f_2) = \mathsf{eval\text{-}expr}\ (t + \mathsf{used\text{-}tmps}\ e_1)\ e_2 \\
 & & \mathtt{in}\ (D_1 \cup D_2,\ \lambda j.\ f\ (f_1\ j)\ (f_2\ j))
\end{array}
$$

Fig. 17: Expression evaluation

the number of temporaries that are necessary to evaluate expression e, where every $\textsc{Mem}$ expression accounts to one temporary. With $\mathsf{issue\text{-}expr}\ t\ e$ we obtain the instruction list for expression e starting at temporary t, whereas $\mathsf{eval\text{-}expr}\ t\ e$ constructs the operation as a pair of the domain and a function on the temporaries.

The program transitions are defined in Figure 18. We instantiate the program state by a tuple $(s,\ t)$ containing the statement s and the temporary counter t. To assign an expression e to an address(-expression) a we first create the memory instructions for evaluation the address a and transforming the expression to an operation on temporaries. The temporary counter is incremented accordingly. When the value is available in the temporaries we continue by creating the memory instructions for evaluation of expression e followed by the corresponding store operation. Note that the ownership annotations can depend on the temporaries and thus can take the calculated address into account.

Execution of compare and swap CAS involves evaluation of three expressions, the address a the compare value c_e and the swap value s_e. It is finally mapped to the read-modify-write instruction RMW of the memory system. Recall that execution of RMW first stores the memory content at address a to the specified temporary. The condition compares this value with the result of evaluating c_e and writes swap value s_a if successful. In either case the temporary finally returns the old value read.

Sequential composition is straightforward. An if-then-else is computed by first issuing the memory instructions for evaluation of condition e and transforming the condition to an operation on temporaries. When the result is available the transition to the first or second statement is made, depending on the result of isTrue. Execution of the loop is defined

$$\frac{\forall\, sop.\ a \neq \textsc{Tmp}\ sop \qquad a' = \textsc{Tmp}\ (\textsf{eval-expr}\ t\ a) \qquad t' = t + \textsf{used-tmps}\ a \qquad is = \textsf{issue-expr}\ t\ a}{j \vdash (\textsc{Assign}\ volatile\ a\ e\ A\ L\ R\ W,\ t) \rightarrow_{\mathsf{p}} ((\textsc{Assign}\ volatile\ a'\ e\ A\ L\ R\ W,\ t'),\ is)}$$

$$\frac{D \subseteq \textsf{dom}\ j \qquad is = \textsf{issue-expr}\ t\ e\ @\ [\textsc{Write}\ volatile\ (a\ j)\ (\textsf{eval-expr}\ t\ e)\ (A\ j)\ (L\ j)\ (R\ j)\ (W\ j)]}{j \vdash (\textsc{Assign}\ volatile\ (\textsc{Tmp}\ (D,\ a))\ e\ A\ L\ R\ W,\ t) \rightarrow_{\mathsf{p}} ((\textsc{Skip},\ t + \textsf{used-tmps}\ e),\ is)}$$

$$\frac{\forall\, sop.\ a \neq \textsc{Tmp}\ sop \qquad a' = \textsc{Tmp}\ (\textsf{eval-expr}\ t\ a) \qquad t' = t + \textsf{used-tmps}\ a \qquad is = \textsf{issue-expr}\ t\ a}{j \vdash (\text{CAS}\ a\ c_{\mathsf{e}}\ s_{\mathsf{e}}\ A\ L\ R\ W,\ t) \rightarrow_{\mathsf{p}} ((\text{CAS}\ a'\ c_{\mathsf{e}}\ s_{\mathsf{e}}\ A\ L\ R\ W,\ t'),\ is)}$$

$$\frac{\forall\, sop.\ c_{\mathsf{e}} \neq \textsc{Tmp}\ sop \qquad c_{\mathsf{e}}' = \textsc{Tmp}\ (\textsf{eval-expr}\ t\ c_{\mathsf{e}}) \qquad t' = t + \textsf{used-tmps}\ c_{\mathsf{e}} \qquad is = \textsf{issue-expr}\ t\ c_{\mathsf{e}}}{j \vdash (\text{CAS}\ (\textsc{Tmp}\ a)\ c_{\mathsf{e}}\ s_{\mathsf{e}}\ A\ L\ R\ W,\ t) \rightarrow_{\mathsf{p}} ((\text{CAS}\ (\textsc{Tmp}\ a)\ c_{\mathsf{e}}'\ s_{\mathsf{e}}\ A\ L\ R\ W,\ t'),\ is)}$$

$$\frac{\begin{array}{c} D_{\mathsf{a}} \subseteq \textsf{dom}\ j \\ D_{\mathsf{c}} \subseteq \textsf{dom}\ j \qquad \textsf{eval-expr}\ t\ s_{\mathsf{e}} = (D,\ f) \qquad t' = t + \textsf{used-tmps}\ s_{\mathsf{e}} \qquad cond = (\lambda j.\ \textsf{the}\ (j\ t') = c\ j) \\ ret = (\lambda v_1\ v_2.\ v_1) \qquad is = \textsf{issue-expr}\ t\ s_{\mathsf{e}}\ @\ [\text{RMW}\ (a\ j)\ t'\ (D,\ f)\ cond\ ret\ (A\ j)\ (L\ j)\ (R\ j)\ (W\ j)] \end{array}}{j \vdash (\text{CAS}\ (\textsc{Tmp}\ (D_{\mathsf{a}},\ a))\ (\textsc{Tmp}\ (D_{\mathsf{c}},\ c))\ s_{\mathsf{e}}\ A\ L\ R\ W,\ t) \rightarrow_{\mathsf{p}} ((\textsc{Skip},\ \textsf{Suc}\ t'),\ is)}$$

$$\frac{j \vdash (s_1,\ t) \rightarrow_{\mathsf{p}} ((s_1',\ t'),\ is)}{j \vdash (\textsc{Seq}\ s_1\ s_2,\ t) \rightarrow_{\mathsf{p}} ((\textsc{Seq}\ s_1'\ s_2,\ t'),\ is)}$$

$$\frac{}{j \vdash (\textsc{Seq}\ \textsc{Skip}\ s_2,\ t) \rightarrow_{\mathsf{p}} ((s_2,\ t),\ [])}$$

$$\frac{\forall\, sop.\ e \neq \textsc{Tmp}\ sop \qquad e' = \textsc{Tmp}\ (\textsf{eval-expr}\ t\ e) \qquad t' = t + \textsf{used-tmps}\ e \qquad is = \textsf{issue-expr}\ t\ e}{j \vdash (\textsc{Cond}\ e\ s_1\ s_2,\ t) \rightarrow_{\mathsf{p}} ((\textsc{Cond}\ e'\ s_1\ s_2,\ t'),\ is)}$$

$$\frac{D \subseteq \textsf{dom}\ j \qquad \textsf{isTrue}\ (e\ j)}{j \vdash (\textsc{Cond}\ (\textsc{Tmp}\ (D,\ e))\ s_1\ s_2,\ t) \rightarrow_{\mathsf{p}} ((s_1,\ t),\ [])}$$

$$\frac{D \subseteq \textsf{dom}\ j \qquad \neg\ \textsf{isTrue}\ (e\ j)}{j \vdash (\textsc{Cond}\ (\textsc{Tmp}\ (D,\ e))\ s_1\ s_2,\ t) \rightarrow_{\mathsf{p}} ((s_2,\ t),\ [])}$$

$$\frac{}{j \vdash (\textsc{While}\ e\ s,\ t) \rightarrow_{\mathsf{p}} ((\textsc{Cond}\ e\ (\textsc{Seq}\ s\ (\textsc{While}\ e\ s))\ \textsc{Skip},\ t),\ [])}$$

$$\frac{}{j \vdash (\text{S}\textsc{Ghost}\ A\ L\ R\ W,\ t) \rightarrow_{\mathsf{p}} ((\textsc{Skip},\ t),\ [\textsc{Ghost}\ (A\ j)\ (L\ j)\ (R\ j)\ (W\ j)])}$$

$$\frac{}{j \vdash (\text{S}\textsc{Fence},\ t) \rightarrow_{\mathsf{p}} ((\textsc{Skip},\ t),\ [\textsc{Fence}])}$$

Fig. 18: Program transitions

by stepwise unfolding. Ghost and fence statements are just propagated to the memory system.

To instantiate Theorem 2 with PIMP we define the invariant parameter *valid*, which has to be maintained by all transitions of PIMP, the memory system and the store buffer. Let j be the valuation of temporaries in the current configuration, for every thread configuration $ts_{\mathsf{sb}[i]} = ((s, t), is, j, sb, \mathcal{D}, \mathcal{O})$ where $i < |ts_{\mathsf{sb}}|$ we require: (i) The domain of all intermediate Tmp (D, f) expressions in statement s is below counter t. (ii) All temporaries in the memory system including the store buffer are below counter t. (iii) All temporaries less than counter t are either already defined in the temporaries j or are outstanding read temporaries in the memory system.

For the PIMP transitions we prove these invariants by rule induction on the semantics. For the memory system (including the store buffer steps) the invariants are straightforward. The memory system does not alter the program state and does not create new temporaries, only the PIMP transitions create new ones in strictly ascending order.

7 Conclusion

We have presented a practical and flexible programming discipline for concurrent programs that ensures sequential consistency on TSO machines, such as present x64 architectures. Our approach covers a wide variety of concurrency control, covering locking, data races, single writer multiple readers, read only and thread local portions of memory. We minimize the need for store buffer flushes to optimize the usage of the hardware. Our theorem is not coupled to a specific logical framework like separation logic but is based on more fundamental arguments, namely the adherence to the programming discipline which can be discharged within any program logic using the standard sequential consistent memory model, without any of the complications of TSO.

Related work. Disclaimer. This contribution presents the state of our work from 2010 [8]. Finally, 8 years later, we made the AFP submission for Isabelle2018. This related work paragraph does not thoroughly cover publications that came up in the meantime.

A categorization of various weak memory models is presented in [2]. It is compatible with the recent revisions of the Intel manuals [10] and the revised x86 model presented in [15]. The state of the art in formal verification of concurrent programs is still based on a sequentially consistent memory model. To justify this on a weak memory model often a quite drastic approach is chosen, allowing only coarse-grained concurrency usually implemented by locking. Thereby data races are ruled out completely and there are results that data race free programs can be considered as sequentially consistent for example for the Java memory model [3, 18] or the x86 memory model [15]. Ridge [17] considers weak memory and data-races and verifies Peterson's mutual exclusion algorithm. He ensures sequentially consistency by flushing after every write to shared memory. Burckhardt and Musuvathi [6] describe an execution monitor that efficiently checks whether a sequentially consistent TSO execution has a single-step extension that is not sequentially consistent. Like our approach, it avoids having to consider the store buffers as an explicit part of the state. However, their condition requires maintaining in ghost state enough history information to determine causality between events, which means maintaining a vector clock (which is itself unbounded) for each memory address. Moreover, causality (being essentially graph reachability) is already not first-order, and hence unsuitable for many types of program verification. Closely related to our work is the draft of Owens [14] which also investigates on the conditions for sequential consistent reasoning within TSO. The notion of a *triangular-race* free trace is established to exactly characterize the traces on a TSO machine that are still sequentially consistent. A triangular race occurs between a read and a write of two different threads to the same address, when the reader still has some outstanding writes in the store buffer. To avoid the triangular race the reader has to flush the store buffer before reading. This is essentially the same condition that our framework enforces, if we limit every address to be unowned and every access to be volatile. We regard this limitation as too strong for practical programs, where non-volatile accesses (without any flushes) to temporarily local portions of memory (e.g. lock protected data) is common practice. This is our core motivation for introducing the ownership based programming discipline. We are aware of two extensions of our work that were published in the meantime. Chen *et al.* [7] also take effects of the MMU into account and generalize our reduction theorem to handle programs that edit page tables. Oberhauser [13] improves on the flushing policy to also take non-triangular races into account and facilitates an alternative proof approach.

Limitations. There is a class of important programs that are not sequentially consistent but nevertheless correct.

First consider a simple spinlock implementation with a volatile lock `l`, where `l == 0` indicates that the lock is not taken. The following code acquires the lock:

```
while(!interlocked_test_and_set(l));
<critical section accessing protected objects>,
```

and with the assignment `l = 0` we can release the lock again. Within our framework address `l` can be considered *unowned* (and hence shared) and every access to it is *volatile*. We do not have to transfer ownership of the lock `l` itself but of the objects it protects. As acquiring the lock is an expensive interlocked oprations anyway there are no additional restrictions from our framework. The interesting point is the release of the lock via the volatile write `l=0`. This leaves the dirty bit set, and hence our programming discipline requires a flushing instruction before the next volatile read. If `l` is the only volatile variable this is fine, since the next operation will be a lock acquire again which is interlocked and thus flushes the store buffer. So there is no need for an additonal fence. But in general this is not the case and we would have to insert a fence after the lock release to make the dirty bit clean again and to stay sequentially consistent. However, can we live without the fence? For the correctness of the mutal-exclusion algorithm we can, but we leave the domain of sequential consistent reasoning. The intuitive reason for correctness is that the threads waiting for the lock do no harm while waiting. They only take some action if they see the lock being zero again, this is when the lock release has made its way out of the store buffer.

Another typical example is the following simplified form of barrier synchronization: each processor has a flag that it writes (with ordinarry volatile writes without any flushing) and other processors read, and each processor waits for all processors to set their flags before continuing past the barrier. This is not sequentially consistent – each processor might see his own flag set and later see all other flags clear – but it is still correct.

Common for these examples is that there is only a single writer to an address, and the values written are monotonic in a sense that allows the readers to draw the correct conlcusion when they observe a certain value. This pattern is named *Publication Idiom* in Owens work [14].

Future work. The first direction of future work is to try to deal with the limitations of sequential consistency described above and try to come up with a more general reduction theorem that can also handle non sequential consistent code portions that follow some monotonicity rules.

Another direction of future work is to take compiler optimization into account. Our volatile accesses correspond roughly to volatile memory accesses within a C program. An optimizing compiler is free to convert any sequence of non-volatile accesses into a (sequentially semantically equivalent) sequence of accesses. As long as execution is sequentially consistent, equivalence of these programs (e.g., with respect to final states of executions that end with volatile operations) follows immediately by reduction. However, some compilers are a little more lenient in their optimizations, and allow operations on certain local variables to move across volatile operations. In the context of C (where pointers to stack variables can be passed by pointer), the notion of "locality" is somewhat tricky, and makes essential use of C forbidding (semantically) address arithmetic across memory objects.

Acknowledgements

We thank Mark Hillebrand for discussions and feedback on this work and extensive comments on this report.

A Appendix

After the explanatory text in the main body of the document we now show the plain theory files.

theory ReduceStoreBuffer
imports Main
begin

A.1 Memory Instructions

type-synonym addr = nat
type-synonym val = nat
type-synonym tmp = nat

type-synonym tmps = tmp ⇒ val option
type-synonym sop = tmp set × (tmps ⇒ val) — domain and function

locale valid-sop =
fixes sop :: sop
assumes valid-sop: $\bigwedge$D f j.
  ⟦sop=(D,f); D ⊆ dom j⟧
  ⟹
  f j = f (j|‘D)

type-synonym memory = addr ⇒ val
type-synonym owns = addr set
type-synonym rels = addr ⇒ bool option
type-synonym shared = addr ⇒ bool option
type-synonym acq = addr set
type-synonym rel = addr set

type-synonym lcl = addr set
type-synonym wrt = addr set
type-synonym cond = tmps ⇒ bool
type-synonym ret = val ⇒ val ⇒ val

datatype instr = Read bool addr tmp
| Write bool addr sop acq lcl rel wrt
| RMW addr tmp sop cond ret acq lcl rel wrt
| Fence
| Ghost acq lcl rel wrt

type-synonym instrs = instr list

type-synonym ('p,'sb,'dirty,'owns,'rels) thread-config =
'p × instrs × tmps × 'sb × 'dirty × 'owns × 'rels
type-synonym ('p,'sb,'dirty,'owns,'rels,'shared) global-config =
('p,'sb,'dirty,'owns,'rels) thread-config list × memory × 'shared

definition owned t = (let (p,instrs,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) = t in $\mathcal{O}$)

lemma owned-simp [simp]: owned (p,instrs,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) = ($\mathcal{O}$)
by (simp add: owned-def)

definition $\mathcal{O}$-sb t = (let (p,instrs,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) = t in ($\mathcal{O}$,sb))

lemma $\mathcal{O}$-sb-simp [simp]: $\mathcal{O}$-sb (p,instrs,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) = ($\mathcal{O}$,sb)
by (simp add: $\mathcal{O}$-sb-def)

definition released t = (let (p,instrs,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) = t in $\mathcal{R}$)

lemma released-simp [simp]: released (p,instrs,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) = ($\mathcal{R}$)
by (simp add: released-def)

lemma list-update-id': v = xs ! i ⟹ xs[i := v] = xs
by simp

lemmas converse-rtranclp-induct5 =
converse-rtranclp-induct [**where** a=(m,sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) **and** b=(m',sb',$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$), split-rule,consumes 1, case-names refl step]

A.2 Abstract Program Semantics

locale memory-system =
fixes
memop-step :: (instrs × tmps × 'sb × memory × 'dirty × 'owns × 'rels × 'shared) ⇒
(instrs × tmps × 'sb × memory × 'dirty × 'owns × 'rels × 'shared) ⇒ bool
(‹- $\rightarrow_{\mathsf{m}}$ -› [60,60] 100) **and**

storebuffer-step:: (memory × ′sb × ′owns × ′rels × ′shared) ⇒ (memory × ′sb × ′owns × ′rels × ′shared) ⇒ bool (‹- $\rightarrow_{sb}$ -› [60,60] 100)

locale program =
fixes
program-step :: tmps ⇒ ′p ⇒ ′p × instrs ⇒ bool (‹-⊢ - $\rightarrow_p$ -› [60,60,60] 100)
— A program only accesses the shared memory indirectly, it can read the temporaries and can output a sequence of memory instructions

locale computation = memory-system + program +
constrains
— The constrains are only used to name the types *′sb* and *′p*
storebuffer-step:: (memory × ′sb × ′owns × ′rels × ′shared) ⇒ (memory × ′sb × ′owns × ′rels × ′shared) ⇒ bool **and**
memop-step ::
(instrs × tmps × ′sb × memory × ′dirty × ′owns × ′rels × ′shared) ⇒
(instrs × tmps × ′sb × memory × ′dirty × ′owns × ′rels × ′shared) ⇒ bool
and
program-step :: tmps ⇒ ′p ⇒ ′p × instrs ⇒ bool
fixes
record :: ′p ⇒ ′p ⇒ instrs ⇒ ′sb ⇒ ′sb
begin

inductive concurrent-step ::
(′p,′sb,′dirty,′owns,′rels,′shared) global-config ⇒ (′p,′sb,′dirty,′owns,′rels,′shared) global-config ⇒ bool
(‹- ⇒ -› [60,60] 100)
where
Program:
⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
j⊢p $\rightarrow_p$ (p′,is′) ⟧ ⟹
(ts,m,$\mathcal{S}$) ⇒ (ts[i:=(p′,is@is′,j,record p p′ is′ sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)],m,$\mathcal{S}$)

| Memop:
⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
(is,j,sb,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_m$ (is′,j′,sb′,m′,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$) ⟧
⟹
(ts,m,$\mathcal{S}$) ⇒ (ts[i:=(p,is′,j′,sb′,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)],m′,$\mathcal{S}'$)

| StoreBuffer:
⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
(m,sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{sb}$ (m′,sb′,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$) ⟧ ⟹
(ts,m,$\mathcal{S}$) ⇒ (ts[i:=(p,is,j,sb′,$\mathcal{D}$,$\mathcal{O}'$,$\mathcal{R}'$)],m′,$\mathcal{S}'$)

definition final:: (′p,′sb,′dirty,′owns,′rels,′shared) global-config ⇒ bool
where
final c = (¬ (∃ c′. c ⇒ c′))

```
lemma store-buffer-steps:
assumes sb-step: storebuffer-step^** (m,sb,𝒪,ℛ,𝒮) (m′,sb′,𝒪′,ℛ′,𝒮′)
shows ⋀ts. i < length ts ⟹ ts!i = (p,is,j,sb,𝒟,𝒪,ℛ) ⟹
        concurrent-step^** (ts,m,𝒮) (ts[i:=(p,is,j,sb′,𝒟,𝒪′,ℛ′)],m′,𝒮′)
using sb-step
proof (induct rule: converse-rtranclp-induct5)
  case refl then show ?case
    by (simp add: list-update-id′)
next
  case (step m sb 𝒪 ℛ 𝒮 m″ sb″ 𝒪″ ℛ″ 𝒮″)
  note i-bound = ‹i < length ts›
  note ts-i = ‹ts ! i = (p, is, j, sb, 𝒟, 𝒪, ℛ)›
  note step = ‹(m, sb,𝒪,ℛ,𝒮) →sb (m″, sb″,𝒪″,ℛ″,𝒮″)›
  let ?ts′ = ts[i := (p, is, j, sb″,𝒟, 𝒪″,ℛ″)]
  from StoreBuffer [OF i-bound ts-i step]
  have (ts, m, 𝒮) ⇒ (?ts′, m″, 𝒮″).
  also
  from i-bound have i-bound′: i < length ?ts′ by simp
  from i-bound have ts′-i: ?ts′!i = (p,is,j,sb″,𝒟,𝒪″,ℛ″)
    by simp
  from step.hyps (3) [OF i-bound′ ts′-i] i-bound
  have concurrent-step** (?ts′, m″, 𝒮″) (ts[i := (p, is, j, sb′,𝒟, 𝒪′,ℛ′)], m′, 𝒮′)
    by (simp)
  finally
  show ?case .
qed

lemma step-preserves-length-ts:
  assumes step: (ts,m,𝒮) ⇒ (ts′,m′,𝒮′)
  shows length ts′ = length ts
using step
apply (cases)
apply auto
done
end

lemmas concurrent-step-cases = computation.concurrent-step.cases
[cases set, consumes 1, case-names Program Memop StoreBuffer]

definition augment-shared:: shared ⇒ addr set ⇒ addr set ⇒ shared (‹- ⊕- -› [61,1000,60]
61)
where
𝒮 ⊕W S ≡ (λa. if a ∈ S then Some (a ∈ W) else 𝒮 a)

definition restrict-shared:: shared ⇒ addr set ⇒ addr set ⇒ shared (‹- ⊖- -› [51,1000,50]
51)
where
𝒮 ⊖A L ≡ (λa. if a ∈ L then None
                else (case 𝒮 a of None ⇒ None
```

| Some writeable ⇒ Some (a ∈ A ∨ writeable)))

definition read-only :: shared ⇒ addr set
where
read-only $\mathcal{S}$ ≡ {a. ($\mathcal{S}$ a = Some False)}

definition shared-le:: shared ⇒ shared ⇒ bool (**infix** ‹$\subseteq_s$› 50)
where
$m_1 \subseteq_s m_2 \equiv m_1 \subseteq_m m_2 \wedge$ read-only $m_1 \subseteq$ read-only m_2

lemma shared-leD: $m_1 \subseteq_s m_2 \Longrightarrow m_1 \subseteq_m m_2 \wedge$ read-only $m_1 \subseteq$ read-only m_2
  by (simp add: shared-le-def)

lemma shared-le-map-le: $m_1 \subseteq_s m_2 \Longrightarrow m_1 \subseteq_m m_2$
  by (simp add: shared-le-def)

lemma shared-le-read-only-le: $m_1 \subseteq_s m_2 \Longrightarrow$ read-only $m_1 \subseteq$ read-only m_2
  by (simp add: shared-le-def)

lemma dom-augment [simp]: dom (m $\oplus_W$ S) = dom m ∪ S
  by (auto simp add: augment-shared-def)

lemma augment-empty [simp]: S $\oplus_x$ {} = S
  by (simp add: augment-shared-def)

lemma inter-neg [simp]: X ∩ − L = X − L
  by blast

lemma dom-restrict-shared [simp]: dom (m $\ominus_A$ L) = dom m − L
  by (auto simp add: restrict-shared-def split: option.splits)

lemma restrict-shared-UNIV [simp]: (m $\ominus_A$ UNIV) = Map.empty
  by (auto simp add: restrict-shared-def split: if-split-asm option.splits)

lemma restrict-shared-empty [simp]: (Map.empty $\ominus_A$ L) = Map.empty
  apply (rule ext)
  by (auto simp add: restrict-shared-def split: if-split-asm option.splits)

lemma restrict-shared-in [simp]: a ∈ L ⟹ (m $\ominus_A$ L) a = None
  by (auto simp add: restrict-shared-def split: if-split-asm option.splits)

lemma restrict-shared-out: a ∉ L ⟹ (m $\ominus_A$ L) a =
  map-option (λwriteable. (a ∈ A ∨ writeable)) (m a)
  by (auto simp add: restrict-shared-def split: if-split-asm option.splits)

lemma restrict-shared-out′[simp]:
  a ∉ L ⟹ m a = Some writeable ⟹ (m $\ominus_A$ L) a = Some (a ∈ A ∨ writeable)
  by (simp add: restrict-shared-out)

lemma augment-mono-map′: A $\subseteq_m$ B $\Longrightarrow$ (A $\oplus_x$ C) $\subseteq_m$ (B $\oplus_x$ C)
by (auto simp add: augment-shared-def map-le-def domIff)

lemma augment-mono-map: A $\subseteq_s$ B $\Longrightarrow$ (A $\oplus_x$ C) $\subseteq_s$ (B $\oplus_x$ C)
by (auto simp add: augment-shared-def shared-le-def map-le-def read-only-def dom-def split: option.splits if-split-asm)

lemma restrict-mono-map: A $\subseteq_s$ B $\Longrightarrow$ (A $\ominus_x$ C) $\subseteq_s$ (B $\ominus_x$ C)
by (auto simp add: restrict-shared-def shared-le-def map-le-def read-only-def dom-def split: option.splits if-split-asm)

lemma augment-mono-aux: dom A $\subseteq$ dom B $\Longrightarrow$ dom (A $\oplus_x$ C) $\subseteq$ dom (B $\oplus_x$ C)
by auto

lemma restrict-mono-aux: dom A $\subseteq$ dom B $\Longrightarrow$ dom (A $\ominus_x$ C) $\subseteq$ dom (B $\ominus_x$ C)
by auto

lemma read-only-mono: S $\subseteq_m$ S′ $\Longrightarrow$ a $\in$ read-only S $\Longrightarrow$ a $\in$ read-only S′
by (auto simp add: map-le-def domIff read-only-def dest!: bspec)

lemma in-read-only-restrict-conv:
a $\in$ read-only ($\mathcal{S}$ $\ominus_A$ L) = (a $\in$ read-only $\mathcal{S}$ $\wedge$ a $\notin$ L $\wedge$ a $\notin$ A)
by (auto simp add: read-only-def restrict-shared-def split: option.splits if-split-asm)

lemma in-read-only-augment-conv: a $\in$ read-only ($\mathcal{S}$ $\oplus_W$ R) = (if a $\in$ R then a $\notin$ W else a $\in$ read-only $\mathcal{S}$)
by (auto simp add: read-only-def augment-shared-def)

lemmas in-read-only-convs = in-read-only-restrict-conv in-read-only-augment-conv

lemma read-only-dom: read-only $\mathcal{S}$ $\subseteq$ dom $\mathcal{S}$
by (auto simp add: read-only-def dom-def)

lemma read-only-empty [simp]: read-only Map.empty = {}
by (auto simp add: read-only-def)

lemma restrict-shared-fuse: S $\ominus_A$ L $\ominus_B$ M = (S $\ominus_{(A \cup B)}$ (L $\cup$ M))
apply (rule ext)
apply (auto simp add: restrict-shared-def split: option.splits if-split-asm)
done

lemma restrict-shared-empty-set [simp]: S $\ominus_{\{\}}$ {} = S
apply (rule ext)
apply (auto simp add: restrict-shared-def split: option.splits if-split-asm)
done

definition augment-rels:: addr set $\Rightarrow$ addr set $\Rightarrow$ rels $\Rightarrow$ rels
where

augment-rels S R $\mathcal{R}$ = (λa. if a $\in$ R
then (case $\mathcal{R}$ a of
None $\Rightarrow$ Some (a $\in$ S)
| Some s $\Rightarrow$ Some (s $\wedge$ (a $\in$ S)))
else $\mathcal{R}$ a)

declare domIff [iff del]

A.3 Memory Transitions

locale gen-direct-memop-step =
fixes emp::$'$rels **and** aug::owns $\Rightarrow$ rel $\Rightarrow$ $'$rels $\Rightarrow$ $'$rels
begin
inductive gen-direct-memop-step :: (instrs $\times$ tmps $\times$ unit $\times$ memory $\times$ bool $\times$ owns $\times$ $'$rels $\times$ shared) $\Rightarrow$
(instrs $\times$ tmps $\times$ unit $\times$ memory $\times$ bool $\times$ owns $\times$ $'$rels $\times$ shared) $\Rightarrow$ bool
(‹- $\rightarrow$ -› [60,60] 100)
where
Read: (Read volatile a t # is,j, x, m,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow$
(is, j (t$\mapsto$m a), x, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$)

| WriteNonVolatile:
(Write False a (D,f) A L R W#is, j, x, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow$
(is, j, x, m(a := f j), $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$)

| WriteVolatile:
(Write True a (D,f) A L R W# is, j, x, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow$
(is, j, x, m(a:=f j), True, $\mathcal{O}$ $\cup$ A $-$ R, emp, $\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)

| Fence:
(Fence # is, j, x, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow$ (is, j,x, m, False, $\mathcal{O}$, emp, $\mathcal{S}$)

| RMWReadOnly:
$[\![$$\neg$ cond (j(t$\mapsto$m a))$]\!]$ $\Longrightarrow$
(RMW a t (D,f) cond ret A L R W # is, j, x, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow$ (is, j(t$\mapsto$m a),x,m, False, $\mathcal{O}$, emp, $\mathcal{S}$)

| RMWWrite:
$[\![$cond (j(t$\mapsto$m a))$]\!]$ $\Longrightarrow$
(RMW a t (D,f) cond ret A L R W# is, j, x, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow$
(is, j(t$\mapsto$ret (m a) (f(j(t$\mapsto$m a)))),x, m(a:= f(j(t$\mapsto$m a))), False,$\mathcal{O}$ $\cup$ A $-$ R, emp, $\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)

| Ghost:
(Ghost A L R W # is, j, x, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow$
(is, j, x, m, $\mathcal{D}$, $\mathcal{O}$ $\cup$ A $-$ R, aug (dom $\mathcal{S}$) R $\mathcal{R}$, $\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
end

interpretation direct-memop-step: gen-direct-memop-step Map.empty augment-rels **.**

term direct-memop-step.gen-direct-memop-step
abbreviation direct-memop-step :: (instrs × tmps × unit × memory × bool × owns × rels × shared) ⇒
(instrs × tmps × unit × memory × bool × owns × rels × shared) ⇒ bool
(‹- → -› [60,60] 100)
where
direct-memop-step ≡ direct-memop-step.gen-direct-memop-step

term x → Y

abbreviation direct-memop-steps ::
(instrs × tmps × unit × memory × bool × owns × rels × shared) ⇒
(instrs × tmps × unit × memory × bool × owns × rels × shared)
⇒ bool
(‹- →* -› [60,60] 100)
where
direct-memop-steps == (direct-memop-step)^**

term x →* Y

interpretation virtual-memop-step: gen-direct-memop-step () (λS R $\mathcal{R}$. ()) **.**

abbreviation virtual-memop-step :: (instrs × tmps × unit × memory × bool × owns × unit × shared) ⇒
(instrs × tmps × unit × memory × bool × owns × unit × shared) ⇒ bool
(‹- $\rightarrow_{\mathsf{v}}$ -› [60,60] 100)
where
virtual-memop-step ≡ virtual-memop-step.gen-direct-memop-step

term x $\rightarrow_{\mathsf{v}}$ Y

abbreviation virtual-memop-steps ::
(instrs × tmps × unit × memory × bool × owns × unit × shared) ⇒
(instrs × tmps × unit × memory × bool × owns × unit × shared)
⇒ bool
(‹- $\rightarrow_{\mathsf{v}}{}^*$ -› [60,60] 100)
where
virtual-memop-steps == (virtual-memop-step)^**

term x →* Y

lemma virtual-memop-step-simulates-direct-memop-step:
assumes step:
(is, j, x, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) → (is′, j′, x′, m′, $\mathcal{D}'$, $\mathcal{O}'$, $\mathcal{R}'$, $\mathcal{S}'$)
shows (is, j, x, m, $\mathcal{D}$, $\mathcal{O}$, (), $\mathcal{S}$) $\rightarrow_{\mathsf{v}}$ (is′, j′, x′, m′, $\mathcal{D}'$, $\mathcal{O}'$, (), $\mathcal{S}'$)
using step
apply (cases)
apply (auto intro: virtual-memop-step.gen-direct-memop-step.intros)

done

A.4 Safe Configurations of Virtual Machines

inductive safe-direct-memop-state :: owns list ⇒ nat ⇒
(instrs × tmps × memory × bool × owns × shared) ⇒ bool
(‹-,-⊢ - $\surd$› [60,60,60] 100)
where
Read: $[\![$a ∈ $\mathcal{O}$ ∨ a ∈ read-only $\mathcal{S}$ ∨ (volatile ∧ a ∈ dom $\mathcal{S}$);
volatile ⟶ ¬ $\mathcal{D}$ $]\!]$
⟹
$\mathcal{O}$s,i⊢(Read volatile a t # is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$

| WriteNonVolatile:
$[\![$a ∈ $\mathcal{O}$; a ∉ dom $\mathcal{S}$$]\!]$
⟹
$\mathcal{O}$s,i⊢(Write False a (D,f) A L R W#is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$

| WriteVolatile:
$[\![$∀j < length $\mathcal{O}$s. i≠j ⟶ a ∉ $\mathcal{O}$s!j;
A ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$; L ⊆ A; R ⊆ $\mathcal{O}$; A ∩ R = {};
∀j < length $\mathcal{O}$s. i≠j ⟶ A ∩ $\mathcal{O}$s!j = {};
a ∉ read-only $\mathcal{S}$$]\!]$
⟹
$\mathcal{O}$s,i⊢(Write True a (D,f) A L R W# is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$

| Fence:
$\mathcal{O}$s,i⊢(Fence # is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$

| Ghost:
$[\![$A ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$; L ⊆ A; R ⊆ $\mathcal{O}$; A ∩ R = {};
∀j < length $\mathcal{O}$s. i≠j ⟶ A ∩ $\mathcal{O}$s!j = {}$]\!]$
⟹
$\mathcal{O}$s,i⊢(Ghost A L R W# is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$

| RMWReadOnly:
$[\![$¬ cond (j(t↦m a)); a ∈ $\mathcal{O}$ ∨ a ∈ dom $\mathcal{S}$$]\!]$ ⟹
$\mathcal{O}$s,i⊢(RMW a t (D,f) cond ret A L R W# is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$

| RMWWrite:
$[\![$cond (j(t↦m a));
∀j < length $\mathcal{O}$s. i≠j ⟶ a ∉ $\mathcal{O}$s!j;
A ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$; L ⊆ A; R ⊆ $\mathcal{O}$; A ∩ R = {};
∀j < length $\mathcal{O}$s. i≠j ⟶ A ∩ $\mathcal{O}$s!j = {};
a ∉ read-only $\mathcal{S}$$]\!]$
⟹
$\mathcal{O}$s,i⊢(RMW a t (D,f) cond ret A L R W# is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$

| Nil: $\mathcal{O}$s,i⊢([], j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$

inductive safe-delayed-direct-memop-state :: owns list $\Rightarrow$ rels list $\Rightarrow$ nat $\Rightarrow$
(instrs $\times$ tmps $\times$ memory $\times$ bool $\times$ owns $\times$ shared) $\Rightarrow$ bool
($\langle$-,-,-$\vdash$ - $\surd\rangle$ [60,60,60,60] 100)
where
Read: $[\![$a $\in \mathcal{O} \vee$ a $\in$ read-only $\mathcal{S} \vee$ (volatile $\wedge$ a $\in$ dom $\mathcal{S}$);
$\forall$j < length $\mathcal{O}$s. i$\neq$j $\longrightarrow$ ($\mathcal{R}$s!j) a $\neq$ Some False; — no release of unshared address
$\neg$ volatile $\longrightarrow$ ($\forall$j < length $\mathcal{O}$s. i$\neq$j $\longrightarrow$ a $\notin$ dom ($\mathcal{R}$s!j));
volatile $\longrightarrow \neg\ \mathcal{D}$ $]\!]$
$\Longrightarrow$
$\mathcal{O}$s,$\mathcal{R}$s,i$\vdash$(Read volatile a t # is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$

| WriteNonVolatile:
$[\![$a $\in \mathcal{O}$; a $\notin$ dom $\mathcal{S}$; $\forall$j < length $\mathcal{O}$s. i$\neq$j $\longrightarrow$ a $\notin$ dom ($\mathcal{R}$s!j)$]\!]$
$\Longrightarrow$
$\mathcal{O}$s,$\mathcal{R}$s,i$\vdash$(Write False a (D,f) A L R W#is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$

| WriteVolatile:
$[\![\forall$j < length $\mathcal{O}$s. i$\neq$j $\longrightarrow$ a $\notin$ ($\mathcal{O}$s!j $\cup$ dom ($\mathcal{R}$s!j));
A $\subseteq$ dom $\mathcal{S} \cup \mathcal{O}$; L $\subseteq$ A; R $\subseteq \mathcal{O}$; A $\cap$ R = {};
$\forall$j < length $\mathcal{O}$s. i$\neq$j $\longrightarrow$ A $\cap$ ($\mathcal{O}$s!j $\cup$ dom ($\mathcal{R}$s!j)) = {};
a $\notin$ read-only $\mathcal{S}]\!]$
$\Longrightarrow$
$\mathcal{O}$s,$\mathcal{R}$s,i$\vdash$(Write True a (D,f) A L R W# is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$

| Fence:
$\mathcal{O}$s,$\mathcal{R}$s,i$\vdash$(Fence # is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$

| Ghost:
$[\![$A $\subseteq$ dom $\mathcal{S} \cup \mathcal{O}$; L $\subseteq$ A; R $\subseteq \mathcal{O}$; A $\cap$ R = {};
$\forall$j < length $\mathcal{O}$s. i$\neq$j $\longrightarrow$ A $\cap$ ($\mathcal{O}$s!j $\cup$ dom ($\mathcal{R}$s!j)) = {}$]\!]$
$\Longrightarrow$
$\mathcal{O}$s,$\mathcal{R}$s,i$\vdash$(Ghost A L R W# is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$

| RMWReadOnly:
$[\![\neg$ cond (j(t$\mapsto$m a)); a $\in \mathcal{O} \vee$ a $\in$ dom $\mathcal{S}$;
$\forall$j < length $\mathcal{O}$s. i$\neq$j $\longrightarrow$ ($\mathcal{R}$s!j) a $\neq$ Some False — no release of unshared address$]\!]$
$\Longrightarrow$
$\mathcal{O}$s,$\mathcal{R}$s,i$\vdash$(RMW a t (D,f) cond ret A L R W# is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$

| RMWWrite:
$[\![$cond (j(t$\mapsto$m a)); a $\in \mathcal{O} \vee$ a $\in$ dom $\mathcal{S}$;
$\forall$j < length $\mathcal{O}$s. i$\neq$j $\longrightarrow$ a $\notin$ ($\mathcal{O}$s!j $\cup$ dom ($\mathcal{R}$s!j));
A $\subseteq$ dom $\mathcal{S} \cup \mathcal{O}$; L $\subseteq$ A; R $\subseteq \mathcal{O}$; A $\cap$ R = {};
$\forall$j < length $\mathcal{O}$s. i$\neq$j $\longrightarrow$ A $\cap$ ($\mathcal{O}$s!j $\cup$ dom ($\mathcal{R}$s!j)) = {};
a $\notin$ read-only $\mathcal{S}]\!]$
$\Longrightarrow$
$\mathcal{O}$s,$\mathcal{R}$s,i$\vdash$(RMW a t (D,f) cond ret A L R W# is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$

| Nil: $\mathcal{O}$s,$\mathcal{R}$s,i$\vdash$([], j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$

lemma memop-safe-delayed-implies-safe-free-flowing:
  assumes safe-delayed: $\mathcal{O}$s,$\mathcal{R}$s,i⊢(is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$
  shows $\mathcal{O}$s,i⊢(is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$
using safe-delayed
proof (cases)
  case Read **thus** ?thesis
    by (fastforce intro!: safe-direct-memop-state.intros)
next
  case WriteNonVolatile **thus** ?thesis
    by (fastforce intro!: safe-direct-memop-state.intros)
next
  case WriteVolatile **thus** ?thesis
    by (fastforce intro!: safe-direct-memop-state.intros)
next
  case Fence **thus** ?thesis
    by (fastforce intro!: safe-direct-memop-state.intros)
next
  case Ghost **thus** ?thesis
  by (fastforce intro!: safe-direct-memop-state.Ghost)
next
  case RMWReadOnly **thus** ?thesis
    by (fastforce intro!: safe-direct-memop-state.intros)
next
  case RMWWrite **thus** ?thesis
    by (fastforce intro!: safe-direct-memop-state.RMWWrite)
next
  case Nil **thus** ?thesis
    by (fastforce intro!: safe-direct-memop-state.Nil)
qed

lemma memop-empty-rels-safe-free-flowing-implies-safe-delayed:
  assumes safe: $\mathcal{O}$s,i⊢(is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$
  assumes empty: $\forall \mathcal{R} \in$ set $\mathcal{R}$s. $\mathcal{R}$ = Map.empty
  assumes leq: length $\mathcal{O}$s = length $\mathcal{R}$s
  assumes unowned-shared: ($\forall$a. ($\forall$i < length $\mathcal{O}$s. a $\notin$ ($\mathcal{O}$s!i)) $\longrightarrow$ a $\in$ dom $\mathcal{S}$)
  assumes Os-i: $\mathcal{O}$s!i = $\mathcal{O}$
  shows $\mathcal{O}$s,$\mathcal{R}$s,i⊢(is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$
using safe
proof (cases)
  case Read **thus** ?thesis
    using leq empty
    by (fastforce intro!: safe-delayed-direct-memop-state.Read dest: nth-mem)
next
  case WriteNonVolatile **thus** ?thesis
    using leq empty
    by (fastforce intro!: safe-delayed-direct-memop-state.intros dest: nth-mem)
next
  case WriteVolatile **thus** ?thesis
  using leq empty
    apply clarsimp

apply (rule safe-delayed-direct-memop-state.WriteVolatile)
apply (auto)
apply (drule nth-mem)
apply fastforce
apply (drule nth-mem)
apply fastforce
done
next
case Fence **thus** ?thesis
by (fastforce intro!: safe-delayed-direct-memop-state.intros)
next
case Ghost **thus** ?thesis
using leq empty
apply clarsimp
apply (rule safe-delayed-direct-memop-state.Ghost)
apply (auto)
apply (drule nth-mem)
apply fastforce
done
next
case RMWReadOnly **thus** ?thesis
using leq empty
by (fastforce intro!: safe-delayed-direct-memop-state.intros dest: nth-mem)
next
case (RMWWrite cond t a A L R D f ret W) **thus** ?thesis
using leq empty unowned-shared [rule-format, **where** a=a] Os-i
apply clarsimp
apply (rule safe-delayed-direct-memop-state.RMWWrite)
apply (auto)
apply (drule nth-mem)
apply fastforce
apply (drule nth-mem)
apply fastforce
done
next
case Nil **thus** ?thesis
by (fastforce intro!: safe-delayed-direct-memop-state.Nil)
qed

inductive id-storebuffer-step::
(memory $\times$ unit $\times$ owns $\times$ rels $\times$ shared) $\Rightarrow$ (memory $\times$ unit $\times$ owns $\times$ rels $\times$ shared) $\Rightarrow$ bool (‹- $\rightarrow_{\mathsf{I}}$ -› [60,60] 100)
where
Id: (m,x,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{I}}$ (m,x,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$)

definition empty-storebuffer-step:: (memory $\times$ 'sb $\times$ 'owns $\times$ 'rels $\times$ 'shared) $\Rightarrow$ (memory $\times$ 'sb $\times$ 'owns $\times$ 'rels $\times$ 'shared) $\Rightarrow$ bool
where
empty-storebuffer-step c c' = False

context program
begin

abbreviation direct-concurrent-step ::
('p,unit,bool,owns,rels,shared) global-config ⇒ ('p,unit,bool,owns,rels,shared) global-config ⇒ bool
(‹- $\Rightarrow_{\mathsf{d}}$ -› [100,60] 100)
where
direct-concurrent-step ≡
computation.concurrent-step direct-memop-step.gen-direct-memop-step empty-storebuffer-step program-step
(λp p$'$ is sb. sb)

abbreviation direct-concurrent-steps::
('p,unit,bool,owns,rels,shared) global-config ⇒ ('p,unit,bool,owns,rels,shared) global-config ⇒ bool
(‹- $\Rightarrow_{\mathsf{d}}{}^*$ -› [60,60] 100)
where
direct-concurrent-steps == direct-concurrent-step^∗∗

abbreviation virtual-concurrent-step ::
('p,unit,bool,owns,unit,shared) global-config ⇒ ('p,unit,bool,owns,unit,shared) global-config ⇒ bool
(‹- $\Rightarrow_{\mathsf{v}}$ -› [100,60] 100)
where
virtual-concurrent-step ≡
computation.concurrent-step virtual-memop-step.gen-direct-memop-step empty-storebuffer-step program-step
(λp p$'$ is sb. sb)

abbreviation virtual-concurrent-steps::
('p,unit,bool,owns,unit,shared) global-config ⇒ ('p,unit,bool,owns,unit,shared) global-config ⇒ bool
(‹- $\Rightarrow_{\mathsf{v}}{}^*$ -› [60,60] 100)
where
virtual-concurrent-steps == virtual-concurrent-step^∗∗

term x $\Rightarrow_{\mathsf{v}}$ Y
term x $\Rightarrow_{\mathsf{d}}$ Y

term x $\Rightarrow_{\mathsf{d}}{}^*$ Y
term x $\Rightarrow_{\mathsf{v}}{}^*$ Y

end

definition
safe-reach step safe cfg ≡
∀ cfg$'$. step^∗∗ cfg cfg$'$ ⟶ safe cfg$'$

lemma safe-reach-safe-refl: safe-reach step safe cfg $\Longrightarrow$ safe cfg
  apply (auto simp add: safe-reach-def)
  done

lemma safe-reach-safe-rtrancl: safe-reach step safe cfg $\Longrightarrow$ step^** cfg cfg′ $\Longrightarrow$ safe cfg′

  by (simp only: safe-reach-def)

lemma safe-reach-steps: safe-reach step safe cfg $\Longrightarrow$ step^** cfg cfg′ $\Longrightarrow$ safe-reach step safe cfg′
  apply (auto simp add: safe-reach-def intro: rtranclp-trans)
  done

lemma safe-reach-step: safe-reach step safe cfg $\Longrightarrow$ step cfg cfg′ $\Longrightarrow$ safe-reach step safe cfg′
  apply (erule safe-reach-steps)
  apply (erule r-into-rtranclp)
  done

context program
begin

abbreviation
 safe-reach-direct ≡ safe-reach direct-concurrent-step

lemma safe-reac-direct-def′:
 safe-reach-direct safe cfg ≡
  ∀ cfg′. cfg $\Rightarrow_{\mathsf{d}}{}^*$ cfg′ ⟶ safe cfg′
  by(simp add: safe-reach-def)

abbreviation
 safe-reach-virtual ≡ safe-reach virtual-concurrent-step

lemma safe-reac-virtual-def′:
 safe-reach-virtual safe cfg ≡
  ∀ cfg′. cfg $\Rightarrow_{\mathsf{v}}{}^*$ cfg′ ⟶ safe cfg′
  by(simp add: safe-reach-def)
end

definition
 safe-free-flowing cfg ≡ let (ts,m,$\mathcal{S}$) = cfg
   in (∀ i < length ts. let (p,is,j,x,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) = ts!i in
    map owned ts,i ⊢(is,j,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{S}$)$\surd$)

lemma safeE: ⟦safe-free-flowing (ts,m,$\mathcal{S}$);i<length ts; ts!i=(p,is,j,x,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)⟧
   $\Longrightarrow$ map owned ts,i ⊢(is,j,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{S}$)$\surd$
  by (auto simp add: safe-free-flowing-def)

definition

safe-delayed cfg ≡ let (ts,m,$\mathcal{S}$) = cfg
in (∀ i < length ts. let (p,is,j,x,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) = ts!i in
map owned ts,map released ts,i ⊢(is,j,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{S}$)$\surd$)

lemma safe-delayedE: ⟦safe-delayed (ts,m,$\mathcal{S}$);i<length ts; ts!i=(p,is,j,x,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)⟧
⟹ map owned ts,map released ts,i ⊢(is,j,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{S}$)$\surd$
by (auto simp add: safe-delayed-def)

definition remove-rels ≡ map (λ(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$). (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,()))

theorem (**in** program) virtual-simulates-direct-step:
assumes step: (ts,m,$\mathcal{S}$) $\Rightarrow_{\mathsf{d}}$ (ts′,m′,$\mathcal{S}'$)
shows (remove-rels ts,m,$\mathcal{S}$) $\Rightarrow_{\mathsf{v}}$ (remove-rels ts′,m′,$\mathcal{S}'$)
using step
proof −
interpret direct-computation:
computation direct-memop-step empty-storebuffer-step program-step λp p′ is sb. sb **.**
interpret virtual-computation:
computation virtual-memop-step empty-storebuffer-step program-step λp p′ is sb. sb **.**
from step **show** ?thesis
proof (cases)
case (Program j p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ p′ is′)
then obtain
ts′: ts′ = ts[j:=(p′,is@is′,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ **and**
m′: m′=m **and**
j-bound: j < length ts **and**
ts-j: ts!j = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) **and**
prog-step: j⊢ p $\rightarrow_{\mathsf{p}}$ (p′, is′)
by auto
from ts-j j-bound **have**
vts-j: remove-rels ts!j = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,()) **by** (auto simp add: remove-rels-def)

from virtual-computation.Program [OF - vts-j prog-step, of m $\mathcal{S}$] j-bound ts′
show ?thesis
by (clarsimp simp add: $\mathcal{S}'$ m′ remove-rels-def map-update)
next
case (Memop j p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ is′ j′ sb′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$)
then obtain
ts′: ts′ = ts[j:=(p,is′,j′,sb′,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)] **and**
j-bound: j < length ts **and**
ts-j: ts!j = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) **and**
mem-step: (is, j, sb, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) → (is′, j′, sb′,m′, $\mathcal{D}'$, $\mathcal{O}'$, $\mathcal{R}'$, $\mathcal{S}'$)
by auto

from ts-j j-bound **have**
vts-j: remove-rels ts!j = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,()) **by** (auto simp add: remove-rels-def)

from virtual-computation.Memop[OF - vts-j virtual-memop-step-simulates-direct-memop-step [OF mem-step]] j-bound ts′

show ?thesis
by (clarsimp simp add: remove-rels-def map-update)
next
case (StoreBuffer - p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ sb′ $\mathcal{O}'$ $\mathcal{R}'$)
hence False
by (auto simp add: empty-storebuffer-step-def)
thus ?thesis **..**
qed
qed

lemmas converse-rtranclp-induct-sbh-steps = converse-rtranclp-induct
[of - (ts,m,$\mathcal{S}$) (ts′,m′,$\mathcal{S}'$), split-rule,
consumes 1, case-names refl step]

theorem (**in** program) virtual-simulates-direct-steps:
assumes steps: (ts,m,$\mathcal{S}$) $\Rightarrow_{\mathsf{d}}{}^*$ (ts′,m′,$\mathcal{S}'$)
shows (remove-rels ts,m,$\mathcal{S}$) $\Rightarrow_{\mathsf{v}}{}^*$ (remove-rels ts′,m′,$\mathcal{S}'$)
using steps
proof (induct rule: converse-rtranclp-induct-sbh-steps)
case refl **thus** ?case **by** auto
next
case (step ts m $\mathcal{S}$ ts″ m″ $\mathcal{S}''$)
then obtain
first: (ts, m, $\mathcal{S}$) $\Rightarrow_{\mathsf{d}}$ (ts″, m″, $\mathcal{S}''$) **and**
hyp: (remove-rels ts″, m″, $\mathcal{S}''$) $\Rightarrow_{\mathsf{v}}{}^*$ (remove-rels ts′, m′, $\mathcal{S}'$)
by blast
note virtual-simulates-direct-step [OF first] **also note** hyp
finally
show ?case **by** blast
qed

locale simple-ownership-distinct =
fixes ts::(′p,′sb,′dirty,owns,′rels) thread-config list
assumes simple-ownership-distinct:
$\bigwedge$i j p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j_i sb_i p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j.
⟦i < length ts; j < length ts; i $\neq$ j;
ts!i = (p_i,is_i,j_i,sb_i,$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$); ts!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
⟧ $\Longrightarrow$ $\mathcal{O}_i \cap \mathcal{O}_j$ = {}

lemma (**in** simple-ownership-distinct)
simple-ownership-distinct-nth-update:
$\bigwedge$i p is j $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ xs sb.
⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
$\forall$j < length ts. i$\neq$j $\longrightarrow$ (let (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$) = ts!j
in ($\mathcal{O}'$) $\cap$ ($\mathcal{O}_j$) ={}) ⟧ $\Longrightarrow$
simple-ownership-distinct (ts[i := (p′,is′,j′,sb′,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)])
apply (unfold-locales)
apply (clarsimp simp add: nth-list-update split: if-split-asm)
apply (force dest: simple-ownership-distinct simp add: Let-def)

apply (fastforce dest: simple-ownership-distinct simp add: Let-def)
apply (fastforce dest: simple-ownership-distinct simp add: Let-def)
done

locale read-only-unowned =
fixes $\mathcal{S}$::shared **and** ts::('p,'sb,'dirty,owns,'rels) thread-config list
assumes read-only-unowned:
$\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
$[\![$i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $]\!]$
$\Longrightarrow$
$\mathcal{O}$ ∩ read-only $\mathcal{S}$ = {}

lemma (**in** read-only-unowned)
read-only-unowned-nth-update:
$\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ acq j sb.
$[\![$i < length ts; $\mathcal{O}$ ∩ read-only $\mathcal{S}$ = {}$]\!]$ $\Longrightarrow$
read-only-unowned $\mathcal{S}$ (ts[i := (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])
apply (unfold-locales)
apply (auto dest: read-only-unowned
simp add: nth-list-update split: if-split-asm)
done

locale unowned-shared =
fixes $\mathcal{S}$::shared **and** ts::('p,'sb,'dirty,owns,'rels) thread-config list
assumes unowned-shared: $- \bigcup$ ((λ(-,-,-,-,-,$\mathcal{O}$,-). $\mathcal{O}$) ‘ set ts) ⊆ dom $\mathcal{S}$

lemma (**in** unowned-shared)
unowned-shared-nth-update:
assumes i-bound: i < length ts
assumes ith: ts!i=(p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
assumes subset: $\mathcal{O}$ ⊆ $\mathcal{O}'$
shows unowned-shared $\mathcal{S}$ (ts[i := (p',is',xs',sb',$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)])
proof −
from i-bound ith subset
have $\bigcup$ ((λ(-,-,-,-,-,$\mathcal{O}$,-). $\mathcal{O}$) ‘ set ts) ⊆
$\bigcup$ ((λ(-,-,-,-,-,$\mathcal{O}$,-). $\mathcal{O}$) ‘ set (ts[i := (p',is',xs',sb',$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)]))

apply (auto simp add: in-set-conv-nth nth-list-update split: if-split-asm)
subgoal for x p'' is'' xs'' sb'' $\mathcal{D}''$ $\mathcal{O}''$ $\mathcal{R}''$ j
apply (case-tac j=i)
apply (rule-tac x=(p',is',xs',sb',$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$) **in** bexI)
apply fastforce
apply (fastforce simp add: in-set-conv-nth)
apply (rule-tac x=(p'',is'',xs'',sb'',$\mathcal{D}''$,$\mathcal{O}''$,$\mathcal{R}''$) **in** bexI)
apply fastforce
apply (fastforce simp add: in-set-conv-nth)
done
done
hence $- \bigcup$ ((λ(-,-,-,-,-,$\mathcal{O}$,-). $\mathcal{O}$) ‘ set (ts[i := (p',is',xs',sb',$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)])) ⊆
$- \bigcup$ ((λ(-,-,-,-,-,$\mathcal{O}$,-). $\mathcal{O}$) ‘ set ts)

```
    by blast
  also note unowned-shared
  finally
  show ?thesis
    by (unfold-locales)
qed

lemma (in unowned-shared) a-unowned-by-others-owned-or-shared:
  assumes i-bound: i < length ts
  assumes ts-i: ts!i = (p,is,j,sb,𝒟,𝒪,ℛ)
  assumes a-unowned-others:
        ∀j<length (map owned ts). i ≠ j ⟶
          (let 𝒪j = (map owned ts)!j in a ∉ 𝒪j)

  shows a ∈ 𝒪 ∨ a ∈ dom 𝒮
proof –
  {
    fix j pj isj 𝒪j ℛj 𝒟j xsj sbj
    assume a-unowned: a ∉ 𝒪
    assume j-bound: j < length ts
    assume jth: ts!j = (pj,isj,xsj, sbj, 𝒟j, 𝒪j,ℛj)
    have a ∉ 𝒪j
    proof (cases i=j)
      case True with a-unowned ts-i jth
      show ?thesis
 by auto
    next
      case False
      from a-unowned-others [rule-format, of j] j-bound jth False
      show ?thesis
 by auto
    qed
  } note lem = this
  {
    assume a ∉ 𝒪
    from lem [OF this]
    have a ∈ – ⋃ ((λ(-,-,-,-,-,𝒪,-). 𝒪) ‘ set ts)
      by (fastforce simp add: in-set-conv-nth)
    with unowned-shared have a ∈ dom 𝒮
      by auto
  }
  then
  show ?thesis
    by auto
qed

lemma (in unowned-shared) unowned-shared′:
  assumes notin: ∀i < length ts. a ∉ owned (ts!i)
  shows a ∈ dom 𝒮
proof –
```

from notin **have** $a \in - \bigcup ((\lambda(\text{-}, \text{-}, \text{-}, \text{-}, \text{-}, \mathcal{O}, \text{-}). \mathcal{O})$ ‘ set ts)
  by (force simp add: in-set-conv-nth)
  with unowned-shared **show** ?thesis **by** blast
qed

lemma unowned-shared-def′: unowned-shared $\mathcal{S}$ ts = (∀a. (∀i < length ts. a ∉ owned (ts!i)) ⟶ a ∈ dom $\mathcal{S}$)
apply rule
apply clarsimp
apply (rule unowned-shared.unowned-shared′)
apply fastforce
apply fastforce
apply (unfold unowned-shared-def)
apply clarsimp
subgoal for x
apply (drule-tac x=x **in** spec)
apply (erule impE)
apply clarsimp
apply (case-tac (ts!i))
apply (drule nth-mem)
apply auto
done
done

definition
initial cfg ≡ let (ts,m,$\mathcal{S}$) = cfg
  in unowned-shared $\mathcal{S}$ ts ∧
    (∀i < length ts. let (p,is,j,x,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) = ts!i in
      $\mathcal{R}$ = Map.empty)

lemma initial-empty-rels: initial (ts,m,$\mathcal{S}$) ⟹ ∀$\mathcal{R}$ ∈ set (map released ts). $\mathcal{R}$ = Map.empty
  by (fastforce simp add: initial-def simp add: in-set-conv-nth)

lemma initial-unowned-shared: initial (ts,m,$\mathcal{S}$) ⟹ unowned-shared $\mathcal{S}$ ts
  by (fastforce simp add: initial-def)

lemma initial-safe-free-flowing-implies-safe-delayed:
assumes init: initial c
assumes safe: safe-free-flowing c
shows safe-delayed c
proof −
  obtain ts $\mathcal{S}$ m **where** c: c=(ts,m,$\mathcal{S}$) **by** (cases c)
  from initial-empty-rels [OF init [simplified c]]
  have rels-empty: ∀$\mathcal{R}$∈set (map released ts). $\mathcal{R}$ = Map.empty**.**
  from initial-unowned-shared [OF init [simplified c]] **have** unowned-shared $\mathcal{S}$ ts
    by auto
  hence us:(∀a. (∀i < length (map owned ts). a ∉ (map owned ts!i)) ⟶ a ∈ dom $\mathcal{S}$)
    by (simp add:unowned-shared-def′)
  {

fix i p is j x $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$
assume i-bound: i < length ts
assume ts-i: ts!i = (p,is,j,x,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
have map owned ts,map released ts,i $\vdash$(is,j,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{S}$)$\surd$
proof –
from safeE [OF safe [simplified c] i-bound ts-i]
have map owned ts,i$\vdash$(is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$**.**
from memop-empty-rels-safe-free-flowing-implies-safe-delayed [OF this rels-empty -us] i-bound ts-i
show ?thesis
by simp
qed
}
then show ?thesis
by (fastforce simp add: c safe-delayed-def)
qed

locale program-progress = program +
assumes progress: j$\vdash$ p $\rightarrow_{\mathsf{p}}$ (p$'$,is$'$) $\Longrightarrow$ p$'$ $\neq$ p $\vee$ is$'$ $\neq$ []

The assumption 'progress' could be avoided if we introduce stuttering steps in lemma undo-local-step or make the scheduling of threads explicit, such that we can directly express that 'thread i does not make a step'.**lemma** (**in** program-progress) undo-local-step:
assumes step: (ts,m,$\mathcal{S}$) $\Rightarrow_{\mathsf{d}}$ (ts$'$,m$'$,$\mathcal{S}'$)
assumes i-bound: i < length ts
assumes unchanged: ts!i = ts$'$!i
assumes safe-delayed-undo: safe-delayed (u-ts,u-m,u-shared) — proof should also work with weaker safe-free-flowing
assumes leq: length u-ts = length ts
assumes others-same: $\forall$j < length ts. j$\neq$i $\longrightarrow$ u-ts!j = ts!j
assumes u-ts-i: u-ts!i=(u-p,u-is,u-tmps,u-x,u-dirty,u-owns,u-rels)
assumes u-m-other: $\forall$a. a $\notin$ u-owns $\longrightarrow$ u-m a = m a
assumes u-m-shared: $\forall$a. a $\in$ u-owns $\longrightarrow$ a $\in$ dom u-shared $\longrightarrow$ u-m a = m a
assumes u-shared: $\forall$a. a $\notin$ u-owns $\longrightarrow$ a $\notin$ owned (ts!i) $\longrightarrow$ u-shared a = $\mathcal{S}$ a
assumes dist: simple-ownership-distinct u-ts
assumes dist-ts: simple-ownership-distinct ts
shows $\exists$u-ts$'$ u-shared$'$ u-m$'$. (u-ts,u-m,u-shared) $\Rightarrow_{\mathsf{d}}$ (u-ts$'$,u-m$'$,u-shared$'$) $\wedge$
— thread i is unchanged
u-ts$'$!i = u-ts!i $\wedge$
($\forall$a $\in$ u-owns. u-shared$'$ a = u-shared a) $\wedge$
($\forall$a $\in$ u-owns. $\mathcal{S}'$ a = $\mathcal{S}$ a) $\wedge$
($\forall$a $\in$ u-owns. u-m$'$ a = u-m a) $\wedge$
($\forall$a $\in$ u-owns. m$'$ a = m a) $\wedge$

— other threads are simulated
($\forall$j < length ts. j$\neq$i $\longrightarrow$ u-ts$'$!j = ts$'$!j) $\wedge$
($\forall$a. a $\notin$ u-owns $\longrightarrow$ a $\notin$ owned (ts!i) $\longrightarrow$ u-shared$'$ a = $\mathcal{S}'$ a) $\wedge$
($\forall$a. a $\notin$ u-owns $\longrightarrow$ u-m$'$ a = m$'$ a)
proof –
interpret direct-computation:

computation direct-memop-step empty-storebuffer-step program-step λp p$'$ is sb. sb **.**
from dist **interpret** simple-ownership-distinct u-ts **.**
from step
show ?thesis
proof (cases)
case (Program j p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ p$'$ is$'$)
then obtain
ts$'$: ts$'$ = ts[j:=(p$'$,is@is$'$,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ **and**
m$'$: m$'$=m **and**
j-bound: j < length ts **and**
ts-j: ts!j = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) **and**
prog-step: j$\vdash$ p $\rightarrow_{\mathsf{p}}$ (p$'$, is$'$)
by auto

from progress [OF prog-step] i-bound unchanged ts-j ts$'$
have neq-j-i: j$\neq$i
by auto

from others-same [rule-format, OF j-bound neq-j-i] ts-j
have u-ts-j: u-ts!j = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by simp
from leq j-bound **have** j-bound$'$: j < length u-ts
by simp
from leq i-bound **have** i-bound$'$: i < length u-ts
by simp

from direct-computation.Program [OF j-bound$'$ u-ts-j prog-step]
have ustep: (u-ts,u-m, u-shared) $\Rightarrow_{\mathsf{d}}$ (u-ts[j := (p$'$, is @ is$'$, j, sb, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$)], u-m, u-shared)**.** **show** ?thesis
apply –
apply (rule exI)
apply (rule exI)
apply (rule exI)
apply (rule conjI)
apply (rule ustep)
using neq-j-i others-same u-m-other u-shared j-bound leq ts-j
apply (auto simp add: nth-list-update ts$'$ $\mathcal{S}'$ m$'$)
done
next
case (Memop j p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ is$'$ j$'$ sb$'$ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$)
then obtain
ts$'$: ts$'$ = ts[j:=(p,is$'$,j$'$,sb$'$,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)] **and**
j-bound: j < length ts **and**
ts-j: ts!j = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) **and**
mem-step: (is, j, sb, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow$ (is$'$, j$'$, sb$'$,m$'$, $\mathcal{D}'$, $\mathcal{O}'$, $\mathcal{R}'$, $\mathcal{S}'$)
by auto

from mem-step i-bound unchanged ts-j

have neq-j-i: j$\neq$i
  by cases (auto simp add: ts$'$)

from others-same [rule-format, OF j-bound neq-j-i] ts-j
have u-ts-j: u-ts!j = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  by simp
from leq j-bound **have** j-bound$'$: j < length u-ts
  by simp
from leq i-bound **have** i-bound$'$: i < length u-ts
  by simp
from safe-delayedE [OF safe-delayed-undo j-bound$'$ u-ts-j]
have safe-j: map owned u-ts,map released u-ts,j$\vdash$(is, j, u-m, $\mathcal{D}$, $\mathcal{O}$, u-shared)$\surd$**.**
from simple-ownership-distinct [OF j-bound$'$ i-bound$'$ neq-j-i u-ts-j u-ts-i]
have owns-u-owns: $\mathcal{O}$ $\cap$ u-owns = {} **.**
from mem-step
show ?thesis
proof (cases)
  case (Read volatile a t)
  then obtain
    is: is = Read volatile a t # is$'$ **and**
    j$'$: j$'$ = j(t $\mapsto$ m a) **and**
    sb$'$: sb$'$=sb **and**
    m$'$: m$'$=m **and**
    $\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**
    $\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
    $\mathcal{R}'$: $\mathcal{R}'$=$\mathcal{R}$ **and**
    $\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$
    by auto
  note eqs$'$ = j$'$ sb$'$ m$'$ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$

  from safe-j [simplified is]
  obtain
    access-cond: a $\in$ $\mathcal{O}$ $\vee$ a $\in$ read-only u-shared $\vee$
              (volatile $\wedge$ a $\in$ dom u-shared)

    and
    clean: volatile $\longrightarrow$ $\neg$ $\mathcal{D}$
    by cases auto
  have u-m-a-eq: u-m a = m a
  proof (cases a $\in$ u-owns)
    case True
    with simple-ownership-distinct [OF j-bound$'$ i-bound$'$ neq-j-i u-ts-j u-ts-i]
    have a $\notin$ $\mathcal{O}$ **by** auto
    with access-cond read-only-dom [of u-shared] **have** a $\in$ dom u-shared
      by auto
    from u-m-shared [rule-format, OF True this]
    show ?thesis **.**
  next
    case False
    from u-m-other [rule-format, OF this]

show ?thesis **.**
qed
note Read′ = direct-memop-step.Read [of volatile a t is′ j sb u-m $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ u-shared]
from direct-computation.Memop [OF j-bound′ u-ts-j, simplified is, OF Read′]
have ustep: (u-ts, u-m, u-shared) $\Rightarrow_{\mathsf{d}}$ (u-ts[j := (p, is′, j(t $\mapsto$ u-m a), sb, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$)], u-m, u-shared)**.**
show ?thesis
apply −
apply (rule exI)
apply (rule exI)
apply (rule exI)
apply (rule conjI)
apply (rule ustep)
using neq-j-i others-same u-m-other u-shared j-bound leq ts-j
by (auto simp add: nth-list-update ts′ eqs′ u-m-a-eq)
next
case (WriteNonVolatile a D f A L R W)
then obtain
is: is = Write False a (D, f) A L R W # is′ **and**
j′: j′ = j **and**
sb′: sb′=sb **and**
m′: m′=m(a:=f j) **and**
$\mathcal{D}$′: $\mathcal{D}$′=$\mathcal{D}$ **and**
$\mathcal{O}$′: $\mathcal{O}$′=$\mathcal{O}$ **and**
$\mathcal{R}$′: $\mathcal{R}$′=$\mathcal{R}$ **and**
$\mathcal{S}$′: $\mathcal{S}$′=$\mathcal{S}$
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}$′ $\mathcal{O}$′ $\mathcal{R}$′ $\mathcal{S}$′

from safe-j [simplified is]
obtain
owned: a $\in$ $\mathcal{O}$ **and** unshared: a $\notin$ dom u-shared
by cases auto

from simple-ownership-distinct [OF j-bound′ i-bound′ neq-j-i u-ts-j u-ts-i] owned
have a-unowned-i: a $\notin$ u-owns
by auto
note Write′ = direct-memop-step.WriteNonVolatile [of a D f A L R W is′ j sb u-m $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ u-shared]
from direct-computation.Memop [OF j-bound′ u-ts-j, simplified is, OF Write′]
have ustep: (u-ts, u-m, u-shared) $\Rightarrow_{\mathsf{d}}$ (u-ts[j := (p, is′, j, sb, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$)], u-m (a := f j), u-shared)**.**
show ?thesis
apply −
apply (rule exI)
apply (rule exI)
apply (rule exI)
apply (rule conjI)
apply (rule ustep)
using neq-j-i others-same u-m-other u-shared j-bound leq ts-j a-unowned-i

apply (auto simp add: nth-list-update ts' eqs')
done
next
case (WriteVolatile a D f A L R W)
then obtain
is: is = Write True a (D, f) A L R W # is' **and**
j': j' = j **and**
sb': sb'=sb **and**
m': m'=m(a:=f j) **and**
$\mathcal{D}'$: $\mathcal{D}'$=True **and**
$\mathcal{O}'$: $\mathcal{O}'=\mathcal{O} \cup A - R$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=Map.empty **and**
$\mathcal{S}'$: $\mathcal{S}'=\mathcal{S} \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L$
by auto
note eqs' = j' sb' m' $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$

from safe-j [simplified is]
obtain
a-unowned-others: $\forall$ k < length u-ts. j$\neq$k $\longrightarrow$ a $\notin$ (map owned u-ts!k $\cup$ dom (map released u-ts!k)) **and**
A: A $\subseteq$ dom u-shared $\cup$ $\mathcal{O}$ **and** L-A: L $\subseteq$ A **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and** A-R: A $\cap$ R = {} **and**
A-unowned-others: $\forall$ k < length u-ts. j$\neq$k $\longrightarrow$ A $\cap$ (map owned u-ts!k $\cup$ dom (map released u-ts!k)) = {} **and**
a-not-ro: a $\notin$ read-only u-shared
by cases auto

note Write' = direct-memop-step.WriteVolatile [of a D f A L R W is' j sb u-m $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ u-shared]
from direct-computation.Memop [OF j-bound' u-ts-j, simplified is, OF Write']
have ustep: (u-ts, u-m, u-shared) $\Rightarrow_{\mathsf{d}}$
(u-ts[j := (p, is', j, sb, True, $\mathcal{O} \cup A - R$, Map.empty)], u-m (a := f j), u-shared $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L).

from A-unowned-others [rule-format, OF i-bound' neq-j-i] u-ts-i i-bound'
have A-u-owns: A $\cap$ u-owns = {} **by** auto
{
fix a
assume a-u-owns: a $\in$ u-owns
have (u-shared $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = u-shared a
using R-owns A-R L-A A-u-owns owns-u-owns a-u-owns
by (auto simp add: restrict-shared-def augment-shared-def split: option.splits)
}
note u-owned-shared = this
from a-unowned-others [rule-format, OF i-bound' neq-j-i] u-ts-i i-bound' **have** a-u-owns: a $\notin$ u-owns **by** auto
{
fix a
assume a-u-owns: a $\notin$ u-owns
assume a-u-owns-orig: a $\notin$ owned (ts!i)

from u-shared [rule-format, OF a-u-owns a-u-owns-orig]
have (u-shared $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a
using R-owns A-R L-A A-u-owns owns-u-owns
by (auto simp add: restrict-shared-def augment-shared-def split: option.splits)
}
note u-unowned-shared = this
{
fix a
assume a-u-owns: a $\in$ u-owns

have ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = $\mathcal{S}$ a
using R-owns A-R L-A A-u-owns owns-u-owns a-u-owns
by (auto simp add: restrict-shared-def augment-shared-def split: option.splits)
}
note $\mathcal{S}'$-shared = this

show ?thesis
apply −
apply (rule exI)
apply (rule exI)
apply (rule exI)
apply (rule conjI)
apply (rule ustep)
using neq-j-i others-same u-m-other u-shared j-bound leq ts-j u-owned-shared a-u-owns u-unowned-shared $\mathcal{S}'$-shared
apply (auto simp add: nth-list-update ts′ eqs′)
done
next
case Fence
then obtain
is: is = Fence # is′ and
j′: j′ = j and
sb′: sb′=sb and
m′: m′=m and
$\mathcal{D}'$: $\mathcal{D}'$=False and
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ and
$\mathcal{R}'$: $\mathcal{R}'$=Map.empty and
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$
note Fence′ = direct-memop-step.Fence [of is′ j sb u-m $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ u-shared]
from direct-computation.Memop [OF j-bound′ u-ts-j, simplified is, OF Fence′]
have ustep: (u-ts, u-m, u-shared) $\Rightarrow_{\mathsf{d}}$ (u-ts[j := (p, is′, j, sb, False, $\mathcal{O}$, Map.empty)], u-m, u-shared).
show ?thesis
apply −
apply (rule exI)
apply (rule exI)
apply (rule exI)
apply (rule conjI)

apply (rule ustep)
using neq-j-i others-same u-m-other u-shared j-bound leq ts-j
by (auto simp add: nth-list-update ts′ eqs′)
next
case (RMWReadOnly cond t a D f ret A L R W)
then obtain
is: is = RMW a t (D, f) cond ret A L R W # is′ **and**
j′: j′ = j(t↦ m a) **and**
sb′: sb′=sb **and**
m′: m′=m **and**
$\mathcal{D}$′: $\mathcal{D}$′=False **and**
$\mathcal{O}$′: $\mathcal{O}$′=$\mathcal{O}$ **and**
$\mathcal{R}$′: $\mathcal{R}$′=Map.empty **and**
$\mathcal{S}$′: $\mathcal{S}$′=$\mathcal{S}$ **and**
cond: ¬ cond (j(t ↦ m a))
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}$′ $\mathcal{O}$′ $\mathcal{R}$′ $\mathcal{S}$′
from safe-j [simplified is] owns-u-owns u-ts-i i-bound′ neq-j-i
obtain
access-cond: a ∉ u-owns ∨ (a ∈ dom u-shared ∧ a ∈ u-owns)
by cases auto

from u-m-other u-m-shared access-cond
have u-m-a-eq: u-m a = m a
by auto
from cond u-m-a-eq **have** cond′: ¬ cond (j(t ↦ u-m a))
by auto
note RMWReadOnly′ = direct-memop-step.RMWReadOnly [of cond j t u-m a D f ret A L R W is′ sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ u-shared,
OF cond′]
from direct-computation.Memop [OF j-bound′ u-ts-j, simplified is, OF RMWReadOnly′]
have ustep: (u-ts, u-m, u-shared) $\Rightarrow_{\mathsf{d}}$ (u-ts[j := (p, is′, j(t ↦ u-m a), sb, False, $\mathcal{O}$, Map.empty)], u-m, u-shared)**.**
show ?thesis
apply −
apply (rule exI)
apply (rule exI)
apply (rule exI)
apply (rule conjI)
apply (rule ustep)
using neq-j-i others-same u-m-other u-shared j-bound leq ts-j
by (auto simp add: nth-list-update ts′ eqs′ u-m-a-eq)
next
case (RMWWrite cond t a D f ret A L R W)
then obtain
is: is = RMW a t (D, f) cond ret A L R W # is′ **and**
j′: j′ = j(t ↦ ret (m a) (f (j(t ↦ m a)))) **and**
sb′: sb′=sb **and**
m′: m′=m(a := f (j(t ↦ m a))) **and**

$\mathcal{D}'$: $\mathcal{D}'$=False **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ ∪ A − R **and**
$\mathcal{R}'$: $\mathcal{R}'$=Map.empty **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L **and**
cond: cond (j(t ↦ m a))
by auto

note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$
from safe-j [simplified is] owns-u-owns u-ts-i i-bound′ neq-j-i
obtain
access-cond: a ∉ u-owns ∨ (a ∈ dom u-shared ∧ a ∈ u-owns)
by cases auto

from u-m-other u-m-shared access-cond
have u-m-a-eq: u-m a = m a
by auto
from cond u-m-a-eq **have** cond′: cond (j(t ↦ u-m a))
by auto
from safe-j [simplified is] cond′
obtain
a-unowned-others: ∀ k < length u-ts. j≠k ⟶ a ∉ (map owned u-ts!k ∪ dom (map released u-ts!k)) **and**
A: A ⊆ dom u-shared ∪ $\mathcal{O}$ **and** L-A: L ⊆ A **and** R-owns: R ⊆ $\mathcal{O}$ **and** A-R: A ∩ R = {} **and**
A-unowned-others: ∀ k < length u-ts. j≠k ⟶ A ∩ (map owned u-ts!k ∪ dom (map released u-ts!k)) = {} **and**
a-not-ro: a ∉ read-only u-shared
by cases auto

note Write′ = direct-memop-step.RMWWrite [of cond j t u-m a D f ret A L R W is′ sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ u-shared,
OF cond′]
from direct-computation.Memop [OF j-bound′ u-ts-j, simplified is, OF Write′]
have ustep: (u-ts, u-m, u-shared) $\Rightarrow_{\mathsf{d}}$
(u-ts[j := (p, is′, j(t ↦ ret (u-m a) (f (j(t ↦ u-m a)))), sb, False, $\mathcal{O}$ ∪ A − R, Map.empty)], u-m(a := f (j(t ↦ u-m a))),
u-shared $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L).

from A-unowned-others [rule-format, OF i-bound′ neq-j-i] u-ts-i i-bound′
have A-u-owns: A ∩ u-owns = {} **by** auto
{
fix a
assume a-u-owns: a ∈ u-owns
have (u-shared $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = u-shared a
using R-owns A-R L-A A-u-owns owns-u-owns a-u-owns
by (auto simp add: restrict-shared-def augment-shared-def split: option.splits)
}
note u-owned-shared = this
from a-unowned-others [rule-format, OF i-bound′ neq-j-i] u-ts-i i-bound′ **have** a-u-owns: a ∉ u-owns **by** auto

{
  fix a
  assume a-u-owns: a $\notin$ u-owns
  assume a-u-owns-orig: a $\notin$ owned (ts!i)
  from u-shared [rule-format, OF a-u-owns this]
  have (u-shared $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a
  using R-owns A-R L-A A-u-owns owns-u-owns
    by (auto simp add: restrict-shared-def augment-shared-def split: option.splits)
}
note u-unowned-shared = this
{
  fix a
  assume a-u-owns: a $\in$ u-owns

  have ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = $\mathcal{S}$ a
  using R-owns A-R L-A A-u-owns owns-u-owns a-u-owns
    by (auto simp add: restrict-shared-def augment-shared-def split: option.splits)
}
note $\mathcal{S}'$-shared = this
show ?thesis
  apply –
  apply (rule exI)
  apply (rule exI)
  apply (rule exI)
  apply (rule conjI)
  apply (rule ustep)
    using neq-j-i others-same u-m-other u-shared j-bound leq ts-j u-owned-shared a-u-owns u-unowned-shared $\mathcal{S}'$-shared
  apply (auto simp add: nth-list-update ts′ eqs′)
  done
next
  case (Ghost A L R W)
  then obtain
    is: is = Ghost A L R W # is′ **and**
    j′: j′ = j **and**
    sb′: sb′=sb **and**
    m′: m′=m **and**
    $\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**
    $\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ $\cup$ A − R **and**
    $\mathcal{R}'$: $\mathcal{R}'$=augment-rels (dom $\mathcal{S}$) R $\mathcal{R}$ **and**
    $\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L
    by auto
  note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$

  from safe-j [simplified is]
  obtain
    A: A $\subseteq$ dom u-shared $\cup$ $\mathcal{O}$ **and** L-A: L $\subseteq$ A **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and** A-R: A $\cap$ R = {} **and**
    A-unowned-others: $\forall$ k < length u-ts. j$\neq$k $\longrightarrow$ A $\cap$ (map owned u-ts!k $\cup$ dom (map released u-ts!k)) = {}

by cases auto

note Ghost$'$ = direct-memop-step.Ghost [of A L R W is$'$ j sb u-m $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ u-shared]
from direct-computation.Memop [OF j-bound$'$ u-ts-j, simplified is, OF Ghost$'$]
have ustep: (u-ts, u-m, u-shared) $\Rightarrow_{\mathsf{d}}$
(u-ts[j := (p, is$'$, j, sb, $\mathcal{D}$, $\mathcal{O}$ ∪ A − R, augment-rels (dom u-shared) R $\mathcal{R}$)], u-m,
u-shared $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L).

from A-unowned-others [rule-format, OF i-bound$'$ neq-j-i] u-ts-i i-bound$'$
have A-u-owns: A ∩ u-owns = {} **by** auto
{
fix a
assume a-u-owns: a ∈ u-owns
have (u-shared $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = u-shared a
using R-owns A-R L-A A-u-owns owns-u-owns a-u-owns
by (auto simp add: restrict-shared-def augment-shared-def split: option.splits)
}
note u-owned-shared = this
{
fix a
assume a-u-owns: a ∉ u-owns
assume a ∉ owned (ts!i)
from u-shared [rule-format, OF a-u-owns this]
have (u-shared $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a
using R-owns A-R L-A A-u-owns owns-u-owns
by (auto simp add: restrict-shared-def augment-shared-def split: option.splits)
}
note u-unowned-shared = this

{
fix a
assume a-u-owns: a ∈ u-owns

have ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = $\mathcal{S}$ a
using R-owns A-R L-A A-u-owns owns-u-owns a-u-owns
by (auto simp add: restrict-shared-def augment-shared-def split: option.splits)
}
note $\mathcal{S}'$-shared = this

from dist-ts
interpret dist-ts-inter: simple-ownership-distinct ts **.**
from dist-ts-inter.simple-ownership-distinct [OF j-bound i-bound neq-j-i ts-j]
have $\mathcal{O}$ ∩ owned (ts!i) = {}
apply (cases ts!i)
apply fastforce+
done

with simple-ownership-distinct [OF j-bound$'$ i-bound$'$ neq-j-i u-ts-j u-ts-i] R-owns u-shared

have augment-eq: augment-rels (dom u-shared) R $\mathcal{R}$ = augment-rels (dom $\mathcal{S}$) R $\mathcal{R}$
  apply –
  apply (rule ext)
  apply (fastforce simp add: augment-rels-def split: option.splits simp add: domIff)

  done

show ?thesis
  apply –
  apply (rule exI)
  apply (rule exI)
  apply (rule exI)
  apply (rule conjI)
  apply (rule ustep)
    using neq-j-i others-same u-m-other u-shared j-bound leq ts-j u-owned-shared u-unowned-shared $\mathcal{S}'$-shared
  apply (auto simp add: nth-list-update ts$'$ eqs$'$ augment-eq)
  done
qed
next
case (StoreBuffer - p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ sb$'$ $\mathcal{O}'$ $\mathcal{R}'$)
hence False
  by (auto simp add: empty-storebuffer-step-def)
thus ?thesis **..**
qed
qed

theorem (**in** program) safe-step-preserves-simple-ownership-distinct:
assumes step: (ts,m,$\mathcal{S}$) $\Rightarrow_{\mathsf{d}}$ (ts$'$,m$'$,$\mathcal{S}'$)
assumes safe: safe-delayed (ts,m,$\mathcal{S}$)
assumes dist: simple-ownership-distinct ts
shows simple-ownership-distinct ts$'$
proof –
interpret direct-computation:
  computation direct-memop-step empty-storebuffer-step program-step λp p$'$ is sb. sb **.**
from dist **interpret** simple-ownership-distinct ts **.**
from step
show ?thesis
proof (cases)
  case (Program j p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ p$'$ is$'$)
  then obtain
    ts$'$: ts$'$ = ts[j:=(p$'$,is@is$'$,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] **and**
    $\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ **and**
    m$'$: m$'$=m **and**
    j-bound: j < length ts **and**
    ts-j: ts!j = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) **and**
    prog-step: j$\vdash$ p $\rightarrow_{\mathsf{p}}$ (p$'$, is$'$)

by auto

from simple-ownership-distinct [OF j-bound - - ts-j]
show simple-ownership-distinct ts′
apply (simp only: ts′)
apply (rule simple-ownership-distinct-nth-update [OF j-bound ts-j])
apply force
done

next
case (Memop j p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ is′ j′ sb′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$)
then obtain
ts′: ts′ = ts[j:=(p,is′,j′,sb′,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)] **and**
j-bound: j < length ts **and**
ts-j: ts!j = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) **and**
mem-step: (is, j, sb, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) → (is′, j′, sb′,m′, $\mathcal{D}'$, $\mathcal{O}'$, $\mathcal{R}'$, $\mathcal{S}'$)
by auto

from safe-delayedE [OF safe j-bound ts-j]
have safe-j: map owned ts,map released ts,j⊢(is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$.
from mem-step
show ?thesis
proof (cases)
case (Read volatile a t)
then obtain
is: is = Read volatile a t # is′ **and**
j′: j′ = j(t ↦ m a) **and**
sb′: sb′=sb **and**
m′: m′=m **and**
$\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=$\mathcal{R}$ **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$

from simple-ownership-distinct [OF j-bound - - ts-j]
show simple-ownership-distinct ts′
apply (simp only: ts′ eqs′)
apply (rule simple-ownership-distinct-nth-update [OF j-bound ts-j])
apply force
done

next
case (WriteNonVolatile a D f A L R W)
then obtain
is: is = Write False a (D, f) A L R W # is′ **and**
j′: j′ = j **and**
sb′: sb′=sb **and**
m′: m′=m(a:=f j) **and**
$\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**

$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=$\mathcal{R}$ **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$
from simple-ownership-distinct [OF j-bound - - ts-j]
show simple-ownership-distinct ts′
apply (simp only: ts′ eqs′)
apply (rule simple-ownership-distinct-nth-update [OF j-bound ts-j])
apply force
done

next
case (WriteVolatile a D f A L R W)
then obtain
is: is = Write True a (D, f) A L R W # is′ **and**
j′: j′ = j **and**
sb′: sb′=sb **and**
m′: m′=m(a:=f j) **and**
$\mathcal{D}'$: $\mathcal{D}'$=True **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ ∪ A − R **and**
$\mathcal{R}'$: $\mathcal{R}'$=Map.empty **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$

from safe-j [simplified is]
obtain
a-unowned-others: ∀k < length ts. j≠k ⟶ a ∉ (map owned ts!k ∪ dom (map released ts!k)) **and**
A: A ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$ **and** L-A: L ⊆ A **and** R-owns: R ⊆ $\mathcal{O}$ **and** A-R: A ∩ R = {} **and**
A-unowned-others: ∀k < length ts. j≠k ⟶ A ∩ (map owned ts!k ∪ dom (map released ts!k)) = {} **and**
a-not-ro: a ∉ read-only $\mathcal{S}$
by cases auto
from simple-ownership-distinct [OF j-bound - - ts-j] R-owns A-R A-unowned-others
show simple-ownership-distinct ts′
apply (simp only: ts′ eqs′)
apply (rule simple-ownership-distinct-nth-update [OF j-bound ts-j])
apply force
done
next
case Fence
then obtain
is: is = Fence # is′ **and**
j′: j′ = j **and**
sb′: sb′=sb **and**
m′: m′=m **and**
$\mathcal{D}'$: $\mathcal{D}'$=False **and**

$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=Map.empty **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$
from simple-ownership-distinct [OF j-bound - - ts-j]
show simple-ownership-distinct ts′
apply (simp only: ts′ eqs′)
apply (rule simple-ownership-distinct-nth-update [OF j-bound ts-j])
apply force
done
next
case (RMWReadOnly cond t a D f ret A L R W)
then obtain
is: is = RMW a t (D, f) cond ret A L R W # is′ **and**
j′: j′ = j(t$\mapsto$ m a) **and**
sb′: sb′=sb **and**
m′: m′=m **and**
$\mathcal{D}'$: $\mathcal{D}'$=False **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=Map.empty **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ **and**
cond: ¬ cond (j(t $\mapsto$ m a))
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$
from simple-ownership-distinct [OF j-bound - - ts-j]
show simple-ownership-distinct ts′
apply (simp only: ts′ eqs′)
apply (rule simple-ownership-distinct-nth-update [OF j-bound ts-j])
apply force
done
next
case (RMWWrite cond t a D f ret A L R W)
then obtain
is: is = RMW a t (D, f) cond ret A L R W # is′ **and**
j′: j′ = j(t $\mapsto$ ret (m a) (f (j(t $\mapsto$ m a)))) **and**
sb′: sb′=sb **and**
m′: m′=m(a := f (j(t $\mapsto$ m a))) **and**
$\mathcal{D}'$: $\mathcal{D}'$=False **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ ∪ A − R **and**
$\mathcal{R}'$: $\mathcal{R}'$=Map.empty **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L **and**
cond: cond (j(t $\mapsto$ m a))
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$
from safe-j [simplified is] cond
obtain
a-unowned-others: ∀k < length ts. j≠k ⟶ a ∉ (map owned ts!k ∪ dom (map released ts!k)) **and**

A: A ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$ **and** L-A: L ⊆ A **and** R-owns: R ⊆ $\mathcal{O}$ **and** A-R: A ∩ R = {} **and**
A-unowned-others: ∀ k < length ts. j≠k ⟶ A ∩ (map owned ts!k ∪ dom (map released ts!k)) = {} **and**
a-not-ro: a ∉ read-only $\mathcal{S}$
by cases auto

from simple-ownership-distinct [OF j-bound - - ts-j] R-owns A-R A-unowned-others
show simple-ownership-distinct ts′
apply (simp only: ts′ eqs′)
apply (rule simple-ownership-distinct-nth-update [OF j-bound ts-j])
apply force
done
next
case (Ghost A L R W)
then obtain
is: is = Ghost A L R W # is′ **and**
j′: j′ = j **and**
sb′: sb′=sb **and**
m′: m′=m **and**
$\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ ∪ A − R **and**
$\mathcal{R}'$: $\mathcal{R}'$=augment-rels (dom $\mathcal{S}$) R $\mathcal{R}$ **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$

from safe-j [simplified is]
obtain
A: A ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$ **and** L-A: L ⊆ A **and** R-owns: R ⊆ $\mathcal{O}$ **and** A-R: A ∩ R = {} **and**
A-unowned-others: ∀ k < length ts. j≠k ⟶ A ∩ (map owned ts!k ∪ dom (map released ts!k)) = {}
by cases auto

from simple-ownership-distinct [OF j-bound - - ts-j] R-owns A-R A-unowned-others
show simple-ownership-distinct ts′
apply (simp only: ts′ eqs′)
apply (rule simple-ownership-distinct-nth-update [OF j-bound ts-j])
apply force
done
qed
next
case (StoreBuffer - p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ sb′ $\mathcal{O}'$ $\mathcal{R}'$)
hence False
by (auto simp add: empty-storebuffer-step-def)
thus ?thesis **..**
qed
qed

theorem (**in** program) safe-step-preserves-read-only-unowned:
  assumes step: $(ts,m,\mathcal{S}) \Rightarrow_{\mathsf{d}} (ts',m',\mathcal{S}')$
  assumes safe: safe-delayed $(ts,m,\mathcal{S})$
  assumes dist: simple-ownership-distinct ts
  assumes ro-unowned: read-only-unowned $\mathcal{S}$ ts
  shows read-only-unowned $\mathcal{S}'$ ts$'$
proof −
  interpret direct-computation:
    computation direct-memop-step empty-storebuffer-step program-step λp p$'$ is sb. sb **.**
  from dist **interpret** simple-ownership-distinct ts **.**
  from ro-unowned **interpret** read-only-unowned $\mathcal{S}$ ts **.**
  from step
  show ?thesis
  proof (cases)
    case (Program j p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ p$'$ is$'$)
    then obtain
      ts$'$: ts$'$ = ts[j:=(p$'$,is@is$'$,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] **and**
      $\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ **and**
      m$'$: m$'$=m **and**
      j-bound: j < length ts **and**
      ts-j: ts!j = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) **and**
      prog-step: $j \vdash p \rightarrow_{\mathsf{p}} (p', is')$
      by auto

    from read-only-unowned [OF j-bound ts-j]
    show read-only-unowned $\mathcal{S}'$ ts$'$
      apply (simp only: ts$'$ $\mathcal{S}'$)
      apply (rule read-only-unowned-nth-update [OF j-bound])
      apply force
      done
  next
    case (Memop j p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ is$'$ j$'$ sb$'$ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$)
    then obtain
      ts$'$: ts$'$ = ts[j:=(p,is$'$,j$'$,sb$'$,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)] **and**
      j-bound: j < length ts **and**
      ts-j: ts!j = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) **and**
      mem-step: $(is, j, sb, m, \mathcal{D}, \mathcal{O}, \mathcal{R}, \mathcal{S}) \rightarrow (is', j', sb', m', \mathcal{D}', \mathcal{O}', \mathcal{R}', \mathcal{S}')$
      by auto

    from safe-delayedE [OF safe j-bound ts-j]
    have safe-j: map owned ts,map released ts,$j \vdash (is, j, m, \mathcal{D}, \mathcal{O}, \mathcal{S})\surd$**.**
    from mem-step
    show ?thesis
    proof (cases)
      case (Read volatile a t)
      then obtain
        is: is = Read volatile a t # is$'$ **and**
        j$'$: j$'$ = j(t $\mapsto$ m a) **and**
        sb$'$: sb$'$=sb **and**
        m$'$: m$'$=m **and**

$\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=$\mathcal{R}$ **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$
by auto
note eqs' = j' sb' m' $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$

from read-only-unowned [OF j-bound ts-j]
show read-only-unowned $\mathcal{S}'$ ts'
apply (simp only: ts' eqs')
apply (rule read-only-unowned-nth-update [OF j-bound])
apply force
done

next
case (WriteNonVolatile a D f A L R W)
then obtain
is: is = Write False a (D, f) A L R W # is' **and**
j': j' = j **and**
sb': sb'=sb **and**
m': m'=m(a:=f j) **and**
$\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=$\mathcal{R}$ **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$
by auto
note eqs' = j' sb' m' $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$
from read-only-unowned [OF j-bound ts-j]
show read-only-unowned $\mathcal{S}'$ ts'
apply (simp only: ts' eqs')
apply (rule read-only-unowned-nth-update [OF j-bound])
apply force
done

next
case (WriteVolatile a D f A L R W)
then obtain
is: is = Write True a (D, f) A L R W # is' **and**
j': j' = j **and**
sb': sb'=sb **and**
m': m'=m(a:=f j) **and**
$\mathcal{D}'$: $\mathcal{D}'$=True **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ ∪ A − R **and**
$\mathcal{R}'$: $\mathcal{R}'$=Map.empty **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L
by auto
note eqs' = j' sb' m' $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$

from safe-j [simplified is]
obtain

a-unowned-others: $\forall$k < length ts. j$\neq$k $\longrightarrow$ a $\notin$ (map owned ts!k $\cup$ dom (map released ts!k)) **and**
A: A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ **and** L-A: L $\subseteq$ A **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and** A-R: A $\cap$ R = {} **and**
A-unowned-others: $\forall$k < length ts. j$\neq$k $\longrightarrow$ A $\cap$ (map owned ts!k $\cup$ dom (map released ts!k)) = {} **and**
a-not-ro: a $\notin$ read-only $\mathcal{S}$
by cases auto

show read-only-unowned $\mathcal{S}'$ ts$'$
proof (unfold-locales)
fix i p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j_i sb_i
assume i-bound: i < length ts$'$
assume ts$'$-i: ts$'$!i = (p_i,is_i,j_i, sb_i, $\mathcal{D}_i$, $\mathcal{O}_i$,$\mathcal{R}_i$)
show $\mathcal{O}_i$ $\cap$ read-only $\mathcal{S}'$ = {}
proof (cases i=j)
case True
with read-only-unowned [OF j-bound ts-j] ts$'$-i A L-A R-owns A-R j-bound
show ?thesis
by (auto simp add: eqs$'$ ts$'$ read-only-def augment-shared-def restrict-shared-def domIff split: option.splits)
next
case False
from simple-ownership-distinct [OF j-bound - False [symmetric] ts-j] ts$'$-i i-bound j-bound False
have $\mathcal{O}$ $\cap$ $\mathcal{O}_i$ = {}
by (fastforce simp add: ts$'$)
with A L-A R-owns A-R j-bound A-unowned-others [rule-format, of i]
read-only-unowned [of i p_i is_i j_i sb_i $\mathcal{D}_i$ $\mathcal{O}_i$ $\mathcal{R}_i$]
False i-bound ts$'$-i False
show ?thesis
by (force simp add: eqs$'$ ts$'$ read-only-def augment-shared-def restrict-shared-def domIff split: option.splits)
qed
qed
next
case Fence
then obtain
is: is = Fence # is$'$ **and**
j$'$: j$'$ = j **and**
sb$'$: sb$'$=sb **and**
m$'$: m$'$=m **and**
$\mathcal{D}'$: $\mathcal{D}'$=False **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=Map.empty **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$
by auto
note eqs$'$ = j$'$ sb$'$ m$'$ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$
from read-only-unowned [OF j-bound ts-j]
show read-only-unowned $\mathcal{S}'$ ts$'$

apply (simp only: ts′ eqs′)
apply (rule read-only-unowned-nth-update [OF j-bound])
apply force
done
next
case (RMWReadOnly cond t a D f ret A L R W)
then obtain
is: is = RMW a t (D, f) cond ret A L R W # is′ **and**
j′: j′ = j(t↦ m a) **and**
sb′: sb′=sb **and**
m′: m′=m **and**
$\mathcal{D}'$: $\mathcal{D}'$=False **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=Map.empty **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ **and**
cond: ¬ cond (j(t ↦ m a))
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$
from read-only-unowned [OF j-bound ts-j]
show read-only-unowned $\mathcal{S}'$ ts′
apply (simp only: ts′ eqs′)
apply (rule read-only-unowned-nth-update [OF j-bound])
apply force
done
next
case (RMWWrite cond t a D f ret A L R W)
then obtain
is: is = RMW a t (D, f) cond ret A L R W # is′ **and**
j′: j′ = j(t ↦ ret (m a) (f (j(t ↦ m a)))) **and**
sb′: sb′=sb **and**
m′: m′=m(a := f (j(t ↦ m a))) **and**
$\mathcal{D}'$: $\mathcal{D}'$=False **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ ∪ A − R **and**
$\mathcal{R}'$: $\mathcal{R}'$=Map.empty **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L **and**
cond: cond (j(t ↦ m a))
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$
from safe-j [simplified is] cond
obtain
a-unowned-others: ∀k < length ts. j≠k ⟶ a ∉ (map owned ts!k ∪ dom (map released ts!k)) **and**
A: A ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$ **and** L-A: L ⊆ A **and** R-owns: R ⊆ $\mathcal{O}$ **and** A-R: A ∩ R = {} **and**
A-unowned-others: ∀k < length ts. j≠k ⟶ A ∩ (map owned ts!k ∪ dom (map released ts!k)) = {} **and**
a-not-ro: a ∉ read-only $\mathcal{S}$
by cases auto

show read-only-unowned $\mathcal{S}'$ ts′

proof (unfold-locales)
fix i p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j_i sb_i
assume i-bound: i < length ts′
assume ts′-i: ts′!i = (p_i,is_i,j_i, sb_i, $\mathcal{D}_i$, $\mathcal{O}_i$,$\mathcal{R}_i$)
show $\mathcal{O}_i$ ∩ read-only $\mathcal{S}'$ = {}
proof (cases i=j)
case True
with read-only-unowned [OF j-bound ts-j] ts′-i A L-A R-owns A-R j-bound
show ?thesis
by (auto simp add: eqs′ ts′ read-only-def augment-shared-def restrict-shared-def domIff split: option.splits)
next
case False
from simple-ownership-distinct [OF j-bound - False [symmetric] ts-j] ts′-i i-bound j-bound False
have $\mathcal{O}$ ∩ $\mathcal{O}_i$ = {}
by (fastforce simp add: ts′)
with A L-A R-owns A-R j-bound A-unowned-others [rule-format, of i]
read-only-unowned [of i p_i is_i j_i sb_i $\mathcal{D}_i$ $\mathcal{O}_i$ $\mathcal{R}_i$]
False i-bound ts′-i False
show ?thesis
by (force simp add: eqs′ ts′ read-only-def augment-shared-def restrict-shared-def domIff split: option.splits)
qed
qed
next
case (Ghost A L R W)
then obtain
is: is = Ghost A L R W # is′ **and**
j′: j′ = j **and**
sb′: sb′=sb **and**
m′: m′=m **and**
$\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ ∪ A − R **and**
$\mathcal{R}'$: $\mathcal{R}'$=augment-rels (dom $\mathcal{S}$) R $\mathcal{R}$ **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$

from safe-j [simplified is]
obtain
A: A ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$ **and** L-A: L ⊆ A **and** R-owns: R ⊆ $\mathcal{O}$ **and** A-R: A ∩ R = {} **and**
A-unowned-others: ∀ k < length ts. j≠k ⟶ A ∩ (map owned ts!k ∪ dom (map released ts!k)) = {}
by cases auto

show read-only-unowned $\mathcal{S}'$ ts′
proof (unfold-locales)
fix i p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j_i sb_i

assume i-bound: i < length ts′
assume ts′-i: ts′!i = (p_i,is_i,j_i, sb_i, $\mathcal{D}_i$, $\mathcal{O}_i$,$\mathcal{R}_i$)
show $\mathcal{O}_i \cap$ read-only $\mathcal{S}'$ = {}
proof (cases i=j)
case True
with read-only-unowned [OF j-bound ts-j] ts′-i A L-A R-owns A-R j-bound
show ?thesis
by (auto simp add: eqs′ ts′ read-only-def augment-shared-def restrict-shared-def domIff split: option.splits)
next
case False
from simple-ownership-distinct [OF j-bound - False [symmetric] ts-j] ts′-i i-bound j-bound False
have $\mathcal{O} \cap \mathcal{O}_i$ = {}
by (fastforce simp add: ts′)
with A L-A R-owns A-R j-bound A-unowned-others [rule-format, of i]
read-only-unowned [of i p_i is_i j_i sb_i $\mathcal{D}_i$ $\mathcal{O}_i$ $\mathcal{R}_i$]
False i-bound ts′-i False
show ?thesis
by (force simp add: eqs′ ts′ read-only-def augment-shared-def restrict-shared-def domIff split: option.splits)
qed
qed
qed
next
case (StoreBuffer - p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ sb′ $\mathcal{O}'$ $\mathcal{R}'$)
hence False
by (auto simp add: empty-storebuffer-step-def)
thus ?thesis **..**
qed
qed

theorem (**in** program) safe-step-preserves-unowned-shared:
assumes step: (ts,m,$\mathcal{S}$) $\Rightarrow_d$ (ts′,m′,$\mathcal{S}'$)
assumes safe: safe-delayed (ts,m,$\mathcal{S}$)
assumes dist: simple-ownership-distinct ts
assumes unowned-shared: unowned-shared $\mathcal{S}$ ts
shows unowned-shared $\mathcal{S}'$ ts′
proof −
interpret direct-computation:
computation direct-memop-step empty-storebuffer-step program-step λp p′ is sb. sb **.**
from dist **interpret** simple-ownership-distinct ts **.**
from unowned-shared **interpret** unowned-shared $\mathcal{S}$ ts **.**
from step
show ?thesis
proof (cases)
case (Program j p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ p′ is′)
then obtain
ts′: ts′ = ts[j:=(p′,is@is′,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] **and**

$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ **and**
m′: m′=m **and**
j-bound: j < length ts **and**
ts-j: ts!j = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) **and**
prog-step: j⊢ p $\rightarrow_{\mathsf{p}}$ (p′, is′)
by auto

show unowned-shared $\mathcal{S}'$ ts′
apply (simp only: ts′ $\mathcal{S}'$)
apply (rule unowned-shared-nth-update [OF j-bound ts-j])
apply force
done
next
case (Memop j p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ is′ j′ sb′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$)
then obtain
ts′: ts′ = ts[j:=(p,is′,j′,sb′,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)] **and**
j-bound: j < length ts **and**
ts-j: ts!j = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) **and**
mem-step: (is, j, sb, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) → (is′, j′, sb′,m′, $\mathcal{D}'$, $\mathcal{O}'$, $\mathcal{R}'$, $\mathcal{S}'$)
by auto

from safe-delayedE [OF safe j-bound ts-j]
have safe-j: map owned ts,map released ts,j⊢(is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$**.**
from mem-step
show ?thesis
proof (cases)
case (Read volatile a t)
then obtain
is: is = Read volatile a t # is′ **and**
j′: j′ = j(t ↦ m a) **and**
sb′: sb′=sb **and**
m′: m′=m **and**
$\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=$\mathcal{R}$ **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$

show unowned-shared $\mathcal{S}'$ ts′
apply (simp only: ts′ eqs′)
apply (rule unowned-shared-nth-update [OF j-bound ts-j])
apply force
done

next
case (WriteNonVolatile a D f A L R W)
then obtain
is: is = Write False a (D, f) A L R W # is′ **and**
j′: j′ = j **and**

sb′: sb′=sb **and**
m′: m′=m(a:=f j) **and**
$\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=$\mathcal{R}$ **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$

show unowned-shared $\mathcal{S}'$ ts′
apply (simp only: ts′ eqs′)
apply (rule unowned-shared-nth-update [OF j-bound ts-j])
apply force
done

next
case (WriteVolatile a D f A L R W)
then obtain
is: is = Write True a (D, f) A L R W # is′ **and**
j′: j′ = j **and**
sb′: sb′=sb **and**
m′: m′=m(a:=f j) **and**
$\mathcal{D}'$: $\mathcal{D}'$=True **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ ∪ A − R **and**
$\mathcal{R}'$: $\mathcal{R}'$=Map.empty **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$

from safe-j [simplified is]
obtain
a-unowned-others: ∀k < length ts. j≠k ⟶ a ∉ (map owned ts!k ∪ dom (map released ts!k)) **and**
A: A ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$ **and** L-A: L ⊆ A **and** R-owns: R ⊆ $\mathcal{O}$ **and** A-R: A ∩ R = {} **and**
A-unowned-others: ∀k < length ts. j≠k ⟶ A ∩ (map owned ts!k ∪ dom (map released ts!k)) = {} **and**
a-not-ro: a ∉ read-only $\mathcal{S}$
by cases auto

show unowned-shared $\mathcal{S}'$ ts′
apply (clarsimp simp add: unowned-shared-def′)
using A R-owns L-A A-R A-unowned-others ts-j j-bound
apply (auto simp add: $\mathcal{S}'$ ts′ $\mathcal{O}'$)
apply (rule unowned-shared′)
apply clarsimp
apply (drule-tac x=i **in** spec)
apply (case-tac i=j)
apply clarsimp

apply clarsimp
apply (drule-tac x=j **in** spec)
apply auto
done
next
case Fence
then obtain
is: is = Fence # is′ **and**
j′: j′ = j **and**
sb′: sb′=sb **and**
m′: m′=m **and**
$\mathcal{D}'$: $\mathcal{D}'$=False **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=Map.empty **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$

show unowned-shared $\mathcal{S}'$ ts′
apply (simp only: ts′ eqs′)
apply (rule unowned-shared-nth-update [OF j-bound ts-j])
apply force
done
next
case (RMWReadOnly cond t a D f ret A L R W)
then obtain
is: is = RMW a t (D, f) cond ret A L R W # is′ **and**
j′: j′ = j(t↦ m a) **and**
sb′: sb′=sb **and**
m′: m′=m **and**
$\mathcal{D}'$: $\mathcal{D}'$=False **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=Map.empty **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ **and**
cond: ¬ cond (j(t ↦ m a))
by auto
note eqs′ = j′ sb′ m′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$
show unowned-shared $\mathcal{S}'$ ts′
apply (simp only: ts′ eqs′)
apply (rule unowned-shared-nth-update [OF j-bound ts-j])
apply force
done
next
case (RMWWrite cond t a D f ret A L R W)
then obtain
is: is = RMW a t (D, f) cond ret A L R W # is′ **and**
j′: j′ = j(t ↦ ret (m a) (f (j(t ↦ m a)))) **and**
sb′: sb′=sb **and**
m′: m′=m(a := f (j(t ↦ m a))) **and**
$\mathcal{D}'$: $\mathcal{D}'$=False **and**

$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ ∪ A − R **and**
$\mathcal{R}'$: $\mathcal{R}'$=Map.empty **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L **and**
cond: cond (j(t ↦ m a))
by auto
note eqs' = j' sb' m' $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$
from safe-j [simplified is] cond
obtain
a-unowned-others: ∀ k < length ts. j≠k ⟶ a ∉ (map owned ts!k ∪ dom (map released ts!k)) **and**
A: A ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$ **and** L-A: L ⊆ A **and** R-owns: R ⊆ $\mathcal{O}$ **and** A-R: A ∩ R = {} **and**
A-unowned-others: ∀ k < length ts. j≠k ⟶ A ∩ (map owned ts!k ∪ dom (map released ts!k)) = {} **and**
a-not-ro: a ∉ read-only $\mathcal{S}$
by cases auto

show unowned-shared $\mathcal{S}'$ ts'
apply (clarsimp simp add: unowned-shared-def')
using A R-owns L-A A-R A-unowned-others ts-j j-bound
apply (auto simp add: $\mathcal{S}'$ ts' $\mathcal{O}'$)
apply (rule unowned-shared')
apply clarsimp
apply (drule-tac x=i **in** spec)
apply (case-tac i=j)
apply clarsimp
apply clarsimp
apply (drule-tac x=j **in** spec)
apply auto
done
next
case (Ghost A L R W)
then obtain
is: is = Ghost A L R W # is' **and**
j': j' = j **and**
sb': sb'=sb **and**
m': m'=m **and**
$\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ ∪ A − R **and**
$\mathcal{R}'$: $\mathcal{R}'$=augment-rels (dom $\mathcal{S}$) R $\mathcal{R}$ **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L
by auto
note eqs' = j' sb' m' $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$

from safe-j [simplified is]
obtain
A: A ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$ **and** L-A: L ⊆ A **and** R-owns: R ⊆ $\mathcal{O}$ **and** A-R: A ∩ R = {} **and**
A-unowned-others: ∀ k < length ts. j≠k ⟶ A ∩ (map owned ts!k ∪ dom (map released ts!k)) = {}

by cases auto
show unowned-shared $\mathcal{S}'$ ts′
apply (clarsimp simp add: unowned-shared-def′)
using A R-owns L-A A-R A-unowned-others ts-j j-bound
apply (auto simp add: $\mathcal{S}'$ ts′ $\mathcal{O}'$)
apply (rule unowned-shared′)
apply clarsimp
apply (drule-tac x=i **in** spec)
apply (case-tac i=j)
apply clarsimp
apply clarsimp
apply (drule-tac x=j **in** spec)
apply auto
done
qed
next
case (StoreBuffer - p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ sb′ $\mathcal{O}'$ $\mathcal{R}'$)
hence False
by (auto simp add: empty-storebuffer-step-def)
thus ?thesis ..
qed
qed

locale program-trace = program +
fixes c — enumeration of configurations: c n $\Rightarrow_{\mathsf{d}}$ c (n + 1) ... $\Rightarrow_{\mathsf{d}}$ c (n + k)
fixes n::nat — starting index
fixes k::nat — steps

assumes step: $\bigwedge$l. l < k $\Longrightarrow$ c (n+l) $\Rightarrow_{\mathsf{d}}$ c (n + (Suc l))

abbreviation (**in** program)
trace ≡ program-trace program-step

lemma (**in** program) trace-0 [simp]: trace c n 0
apply (unfold-locales)
apply auto
done

lemma split-less-Suc: (∀x<Suc k. P x) = (P k ∧ (∀x<k. P x))
apply rule
apply clarsimp
apply clarsimp
apply (case-tac x = k)
apply auto
done

lemma split-le-Suc: (∀x≤Suc k. P x) = (P (Suc k) ∧ (∀x≤k. P x))
apply rule
apply clarsimp
apply clarsimp

```
  apply (case-tac x = Suc k)
  apply auto
  done

lemma (in program) steps-to-trace:
assumes steps: x ⇒d* y
shows ∃c k. trace c 0 k ∧ c 0 = x ∧ c k = y
using steps
proof (induct)
  case base
  thus ?case
    apply (rule-tac x=λk. x in exI)
    apply (rule-tac x=0 in exI)
    by (auto simp add: program-trace-def)
next
  case (step y z)
  have first: x ⇒d* y by fact
  have last: y ⇒d z by fact
  from step.hyps obtain c k where
    trace: trace c 0 k and c-0: c 0 = x and c-k: c k = y
    by auto
  define c' where c' == λi. (if i ≤ k then c i else z)
  from trace last c-k have trace c' 0 (k + 1)
    apply (clarsimp simp add: c'-def program-trace-def)
    apply (subgoal-tac l=k)
    apply  (simp)
    apply (simp)
    done
  with c-0
  show ?case
    apply −
    apply (rule-tac x=c' in exI)
    apply (rule-tac x=k + 1 in exI)
    apply (auto simp add: c'-def)
    done
qed


lemma (in program) trace-preserves-length-ts:
  ⋀l x. trace c n k ⟹ l ≤ k ⟹ x ≤ k ⟹ length (fst (c (n + l))) = length (fst (c (n +
x)))
proof (induct k)
  case 0
  thus ?case by auto
next
  case (Suc k)
  then obtain trace-suc: trace c n (Suc k) and
    l-suc: l ≤ Suc k and
    x-suc: x ≤ Suc k
    by simp
```

interpret direct-computation:
computation direct-memop-step empty-storebuffer-step program-step λp p′ is sb. sb **.**

from trace-suc **obtain**
trace-k: trace c n k **and**
last-step: c (n + k) $\Rightarrow_{\mathsf{d}}$ c (n + (Suc k))
by (clarsimp simp add: program-trace-def)
obtain ts $\mathcal{S}$ m **where** c-k: c (n + k) = (ts, m, $\mathcal{S}$) **by** (cases c (n + k))
obtain ts′ $\mathcal{S}'$ m′ **where** c-suc-k: c (n + (Suc k)) = (ts′, m′, $\mathcal{S}'$) **by** (cases c (n + (Suc k)))
from direct-computation.step-preserves-length-ts [OF last-step [simplified c-k c-suc-k]] c-k c-suc-k
have leq: length (fst (c (n + Suc k))) = length (fst (c (n + k)))
by simp

show ?case
proof (cases l = Suc k)
case True
note l-suc = this
show ?thesis
proof (cases x = Suc k)
case True **with** l-suc **show** ?thesis **by** simp
next
case False
with x-suc **have** x ≤ k **by** simp
from Suc.hyps [OF trace-k this, of k]
have length (fst (c (n + x))) = length (fst (c (n + k)))
by simp
with leq **show** ?thesis **using** l-suc **by** simp
qed
next
case False
with l-suc **have** l-k: l ≤ k
by auto
show ?thesis
proof (cases x = Suc k)
case True
from Suc.hyps [OF trace-k l-k, of k]
have length (fst (c (n + l))) = length (fst (c (n + k))) **by** simp
with leq True **show** ?thesis **by** simp
next
case False
with x-suc **have** x ≤ k **by** simp
from Suc.hyps [OF trace-k l-k this]
show ?thesis **by** simp
qed
qed
qed

lemma (**in** program) trace-preserves-simple-ownership-distinct:

assumes dist: simple-ownership-distinct (fst (c n))
shows $\bigwedge$l. trace c n k $\Longrightarrow$ ($\forall$x < k. safe-delayed (c (n + x))) $\Longrightarrow$
l $\leq$ k $\Longrightarrow$ simple-ownership-distinct (fst (c (n + l)))
proof (induct k)
case 0 **thus** ?case **using** dist **by** auto
next
case (Suc k)
then obtain
trace-suc: trace c n (Suc k) **and**
safe-suc: $\forall$x<Suc k. safe-delayed (c (n + x)) **and**
l-suc: l $\leq$ Suc k
by simp

from trace-suc **obtain**
trace-k: trace c n k **and**
last-step: c (n + k) $\Rightarrow_{\mathsf{d}}$ c (n + (Suc k))
by (clarsimp simp add: program-trace-def)

obtain ts $\mathcal{S}$ m **where** c-k: c (n + k) = (ts, m, $\mathcal{S}$) **by** (cases c (n + k))
obtain ts′ $\mathcal{S}'$ m′ **where** c-suc-k: c (n + (Suc k)) = (ts′, m′, $\mathcal{S}'$) **by** (cases c (n + (Suc k)))

from safe-suc c-suc-k c-k
obtain
safe-up-k: $\forall$x<k. safe-delayed (c (n + x)) **and**
safe-k: safe-delayed (ts,m,$\mathcal{S}$)
by (auto simp add: split-le-Suc)
from Suc.hyps [OF trace-k safe-up-k]
have hyp: $\forall$l $\leq$ k. simple-ownership-distinct (fst (c (n + l)))
by simp

from Suc.hyps [OF trace-k safe-up-k, of k] c-k
have simple-ownership-distinct ts
by simp

from safe-step-preserves-simple-ownership-distinct [OF last-step[simplified c-k c-suc-k] safe-k this]
have simple-ownership-distinct ts′**.**
then show ?case
using c-suc-k hyp l-suc
apply (cases l=Suc k)
apply (auto simp add: split-less-Suc)
done
qed

lemma (**in** program) trace-preserves-read-only-unowned:
assumes dist: simple-ownership-distinct (fst (c n))
assumes ro: read-only-unowned (snd (snd (c n))) (fst (c n))
shows $\bigwedge$l. trace c n k $\Longrightarrow$ ($\forall$x < k. safe-delayed (c (n + x))) $\Longrightarrow$
l $\leq$ k $\Longrightarrow$ read-only-unowned (snd (snd (c (n + l)))) (fst (c (n + l)))

proof (induct k)
  case 0 **thus** ?case **using** ro **by** auto
next
  case (Suc k)
  then obtain
    trace-suc: trace c n (Suc k) **and**
    safe-suc: $\forall$x<Suc k. safe-delayed (c (n + x)) **and**
    l-suc: l $\leq$ Suc k
    by simp

  from trace-suc **obtain**
    trace-k: trace c n k **and**
    last-step: c (n + k) $\Rightarrow_{\mathsf{d}}$ c (n + (Suc k))
    by (clarsimp simp add: program-trace-def)

  obtain ts $\mathcal{S}$ m **where** c-k: c (n + k) = (ts, m, $\mathcal{S}$) **by** (cases c (n + k))
  obtain ts′ $\mathcal{S}'$ m′ **where** c-suc-k: c (n + (Suc k)) = (ts′, m′, $\mathcal{S}'$) **by** (cases c (n + (Suc k)))

  from safe-suc c-suc-k c-k
  obtain
    safe-up-k: $\forall$x<k. safe-delayed (c (n + x)) **and**
    safe-k: safe-delayed (ts,m,$\mathcal{S}$)
    by (auto simp add: split-le-Suc)
  from Suc.hyps [OF trace-k safe-up-k]
  have hyp: $\forall$l $\leq$ k. read-only-unowned (snd (snd (c (n + l)))) (fst (c (n + l)))
    by simp

  from Suc.hyps [OF trace-k safe-up-k, of k] c-k
  have ro′: read-only-unowned $\mathcal{S}$ ts
    by simp

  from trace-preserves-simple-ownership-distinct [**where** c=c **and** n=n, OF dist trace-k safe-up-k, of k] c-k
  have dist′: simple-ownership-distinct ts **by** simp

  from safe-step-preserves-read-only-unowned [OF last-step[simplified c-k c-suc-k] safe-k dist′ ro′]
  have read-only-unowned $\mathcal{S}'$ ts′**.**
  then show ?case
  using c-suc-k hyp l-suc
    apply (cases l=Suc k)
    apply (auto simp add: split-less-Suc)
    done
qed

lemma (**in** program) trace-preserves-unowned-shared:
  assumes dist: simple-ownership-distinct (fst (c n))
  assumes ro: unowned-shared (snd (snd (c n))) (fst (c n))
  shows $\bigwedge$l. trace c n k $\Longrightarrow$ ($\forall$x < k. safe-delayed (c (n + x))) $\Longrightarrow$

$l \leq k \Longrightarrow$ unowned-shared (snd (snd (c (n + l)))) (fst (c (n + l)))
proof (induct k)
  case 0 **thus** ?case **using** ro **by** auto
next
  case (Suc k)
  then obtain
    trace-suc: trace c n (Suc k) **and**
    safe-suc: $\forall$x<Suc k. safe-delayed (c (n + x)) **and**
    l-suc: $l \leq$ Suc k
    by simp

  from trace-suc **obtain**
    trace-k: trace c n k **and**
    last-step: c (n + k) $\Rightarrow_{\mathsf{d}}$ c (n + (Suc k))
    by (clarsimp simp add: program-trace-def)

  obtain ts $\mathcal{S}$ m **where** c-k: c (n + k) = (ts, m, $\mathcal{S}$) **by** (cases c (n + k))
  obtain ts′ $\mathcal{S}'$ m′ **where** c-suc-k: c (n + (Suc k)) = (ts′, m′, $\mathcal{S}'$) **by** (cases c (n + (Suc k)))

  from safe-suc c-suc-k c-k
  obtain
    safe-up-k: $\forall$x<k. safe-delayed (c (n + x)) **and**
    safe-k: safe-delayed (ts,m,$\mathcal{S}$)
    by (auto simp add: split-le-Suc)
  from Suc.hyps [OF trace-k safe-up-k]
  have hyp: $\forall l \leq k$. unowned-shared (snd (snd (c (n + l)))) (fst (c (n + l)))
    by simp

  from Suc.hyps [OF trace-k safe-up-k, of k] c-k
  have ro′: unowned-shared $\mathcal{S}$ ts
    by simp

  from trace-preserves-simple-ownership-distinct [**where** c=c **and** n=n, OF dist trace-k safe-up-k, of k] c-k
  have dist′: simple-ownership-distinct ts **by** simp

  from safe-step-preserves-unowned-shared [OF last-step[simplified c-k c-suc-k] safe-k dist′ ro′]
  have unowned-shared $\mathcal{S}'$ ts′**.**
  then show ?case
  using c-suc-k hyp l-suc
    apply (cases l=Suc k)
    apply (auto simp add: split-less-Suc)
    done
qed

theorem (**in** program-progress) undo-local-steps:
assumes steps: trace c n k

assumes c-n: c n = (ts,m,$\mathcal{S}$)
assumes unchanged: $\forall$l $\leq$ k. ($\forall$ts$_l$ $\mathcal{S}_l$ m$_l$. c (n + l) = (ts$_l$,m$_l$,$\mathcal{S}_l$) $\longrightarrow$ ts$_l$!i=ts!i)
assumes safe: safe-delayed (u-ts, u-m, u-shared)
assumes leq: length u-ts = length ts
assumes i-bound: i < length ts
assumes others-same: $\forall$j < length ts. j$\neq$i $\longrightarrow$ u-ts!j = ts!j
assumes u-ts-i: u-ts!i=(u-p,u-is,u-tmps,u-sb,u-dirty,u-owns,u-rels)
assumes u-m-other: $\forall$a. a $\notin$ u-owns $\longrightarrow$ u-m a = m a
assumes u-m-shared: $\forall$a. a $\in$ u-owns $\longrightarrow$ a $\in$ dom u-shared $\longrightarrow$ u-m a = m a
assumes u-shared: $\forall$a. a $\notin$ u-owns $\longrightarrow$ a $\notin$ owned (ts!i) $\longrightarrow$ u-shared a = $\mathcal{S}$ a
assumes dist: simple-ownership-distinct u-ts
assumes dist-ts: simple-ownership-distinct ts
assumes safe-orig: $\forall$x. x < k $\longrightarrow$ safe-delayed (c (n + x))
shows $\exists$c$'$ l. l $\leq$ k $\wedge$ trace c$'$ n l $\wedge$

c$'$ n = (u-ts, u-m, u-shared) $\wedge$
($\forall$x $\leq$ l. length (fst (c$'$ (n + x))) = length (fst (c (n + x)))) $\wedge$

($\forall$x < l. safe-delayed (c$'$ (n + x))) $\wedge$
(l < k $\longrightarrow$ $\neg$ safe-delayed (c$'$ (n + l))) $\wedge$

($\forall$x $\leq$ l. $\forall$ts$_x$ $\mathcal{S}_x$ m$_x$ ts$_x$$'$ $\mathcal{S}_x$$'$ m$_x$$'$. c (n + x) = (ts$_x$,m$_x$,$\mathcal{S}_x$) $\longrightarrow$ c$'$ (n+ x) = (ts$_x$$'$,m$_x$$'$,$\mathcal{S}_x$$'$) $\longrightarrow$
ts$_x$$'$!i=u-ts!i $\wedge$
($\forall$a $\in$ u-owns. $\mathcal{S}_x$$'$ a = u-shared a) $\wedge$
($\forall$a $\in$ u-owns. $\mathcal{S}_x$ a = $\mathcal{S}$ a) $\wedge$
($\forall$a $\in$ u-owns. m$_x$$'$ a = u-m a) $\wedge$
($\forall$a $\in$ u-owns. m$_x$ a = m a)) $\wedge$

($\forall$x $\leq$ l. $\forall$ts$_x$ $\mathcal{S}_x$ m$_x$ ts$_x$$'$ $\mathcal{S}_x$$'$ m$_x$$'$. c (n + x) = (ts$_x$,m$_x$,$\mathcal{S}_x$) $\longrightarrow$ c$'$ (n + x) = (ts$_x$$'$,m$_x$$'$,$\mathcal{S}_x$$'$) $\longrightarrow$
($\forall$j < length ts$_x$. j$\neq$i $\longrightarrow$ ts$_x$$'$!j = ts$_x$!j) $\wedge$
($\forall$a. a $\notin$ u-owns $\longrightarrow$ a $\notin$ owned (ts!i) $\longrightarrow$ $\mathcal{S}_x$$'$ a = $\mathcal{S}_x$ a) $\wedge$
($\forall$a. a $\notin$ u-owns $\longrightarrow$ m$_x$$'$ a = m$_x$ a))

using steps unchanged safe-orig
proof (induct k)
case 0
show ?case
apply (rule-tac x=λ l. (u-ts, u-m, u-shared) **in** exI)
apply (rule-tac x=0 **in** exI)
thm c-n
apply (simp add: c-n)
apply (clarsimp simp add: 0 leq others-same u-m-other u-shared)
done
next
case (Suc k)
then obtain
trace-suc: trace c n (Suc k) **and**
unchanged-suc: $\forall$l$\leq$Suc k. $\forall$ts$_l$ $\mathcal{S}_l$ m$_l$. c (n + l) = (ts$_l$, m$_l$, $\mathcal{S}_l$) $\longrightarrow$ ts$_l$! i = ts ! i **and**

safe-orig: $\forall$x<k. safe-delayed (c (n + x))
by simp

interpret direct-computation:
computation direct-memop-step empty-storebuffer-step program-step λp p$'$ is sb. sb **.**
from trace-suc **obtain**
trace-k: trace c n k **and**
last-step: c (n + k) $\Rightarrow_{\mathsf{d}}$ c (n + (Suc k))
by (clarsimp simp add: program-trace-def)

from unchanged-suc **obtain**
unchanged-k: $\forall$l$\leq$k. $\forall$ts$_{\mathsf{l}}$ $\mathcal{S}_{\mathsf{l}}$ m$_{\mathsf{l}}$. c (n + l) = (ts$_{\mathsf{l}}$, m$_{\mathsf{l}}$, $\mathcal{S}_{\mathsf{l}}$) $\longrightarrow$ ts$_{\mathsf{l}}$! i = ts ! i **and**
unchanged-suc-k: $\forall$ts$_{\mathsf{l}}$ $\mathcal{S}_{\mathsf{l}}$ m$_{\mathsf{l}}$. c (n + (Suc k)) = (ts$_{\mathsf{l}}$, m$_{\mathsf{l}}$, $\mathcal{S}_{\mathsf{l}}$) $\longrightarrow$ ts$_{\mathsf{l}}$! i = ts ! i
apply −
apply (rule that)
apply auto
apply (drule-tac x=l **in** spec)
apply simp
done

from Suc.hyps [OF trace-k unchanged-k safe-orig] **obtain** c$'$ l **where**
l-k: l $\leq$ k **and**
trace-c$'$-l: trace c$'$ n l **and**
safe-l: ($\forall$x<l. safe-delayed (c$'$ (n + x))) **and**
unsafe-l: (l < k $\longrightarrow$ $\neg$ safe-delayed (c$'$ (n + l))) **and**
c$'$-n: c$'$ n = (u-ts, u-m, u-shared) **and**
leq-l: ($\forall$x$\leq$l. length (fst (c$'$ (n + x))) = length (fst (c (n + x)))) **and**
unchanged-i: ($\forall$x$\leq$l. $\forall$ts$_{\mathsf{x}}$ $\mathcal{S}_{\mathsf{x}}$ m$_{\mathsf{x}}$ ts$_{\mathsf{x}}'$ $\mathcal{S}_{\mathsf{x}}'$ m$_{\mathsf{x}}'$.
c (n + x) = (ts$_{\mathsf{x}}$, m$_{\mathsf{x}}$, $\mathcal{S}_{\mathsf{x}}$) $\longrightarrow$
c$'$ (n + x) = (ts$_{\mathsf{x}}'$, m$_{\mathsf{x}}'$, $\mathcal{S}_{\mathsf{x}}'$) $\longrightarrow$
ts$_{\mathsf{x}}'$! i = u-ts ! i $\wedge$
($\forall$a$\in$u-owns. $\mathcal{S}_{\mathsf{x}}'$ a = u-shared a) $\wedge$
($\forall$a$\in$u-owns. $\mathcal{S}_{\mathsf{x}}$ a = $\mathcal{S}$ a) $\wedge$
($\forall$a$\in$u-owns. m$_{\mathsf{x}}'$ a = u-m a) $\wedge$
($\forall$a$\in$u-owns. m$_{\mathsf{x}}$ a = m a)) **and**
sim: ($\forall$x$\leq$l. $\forall$ts$_{\mathsf{x}}$ $\mathcal{S}_{\mathsf{x}}$ m$_{\mathsf{x}}$ ts$_{\mathsf{x}}'$ $\mathcal{S}_{\mathsf{x}}'$ m$_{\mathsf{x}}'$.
c (n + x) = (ts$_{\mathsf{x}}$, m$_{\mathsf{x}}$, $\mathcal{S}_{\mathsf{x}}$) $\longrightarrow$
c$'$ (n + x) = (ts$_{\mathsf{x}}'$, m$_{\mathsf{x}}'$, $\mathcal{S}_{\mathsf{x}}'$) $\longrightarrow$
($\forall$j<length ts$_{\mathsf{x}}$. j $\neq$ i $\longrightarrow$ ts$_{\mathsf{x}}'$! j = ts$_{\mathsf{x}}$! j) $\wedge$
($\forall$a. a $\notin$ u-owns $\longrightarrow$ a $\notin$ owned (ts!i) $\longrightarrow$ $\mathcal{S}_{\mathsf{x}}'$ a = $\mathcal{S}_{\mathsf{x}}$ a) $\wedge$
($\forall$a. a $\notin$ u-owns $\longrightarrow$ m$_{\mathsf{x}}'$ a = m$_{\mathsf{x}}$ a))
by auto
show ?case
proof (cases l < k)
case True
with True trace-c$'$-l safe-l unsafe-l unchanged-i sim leq-l c$'$-n
show ?thesis
apply −
apply (rule-tac x=c$'$ **in** exI)

```
    apply (rule-tac x=l in exI)
    apply auto
    done
next
  case False
  with l-k have l-k: l=k by auto
  show ?thesis
  proof (cases safe-delayed (c′ (n + k)))
    case False
    with False l-k trace-c′-l safe-l unsafe-l unchanged-i sim leq-l c′-n
    show ?thesis
      apply −
      apply (rule-tac x=c′ in exI)
      apply (rule-tac x=k in exI)
      apply auto
      done
  next
    case True
    note safe-k = this

    obtain tsₖ 𝒮ₖ mₖ where c-k: c (n + k) = (tsₖ,mₖ,𝒮ₖ)
      by (cases c (n + k))

    obtain tsₖ′ 𝒮ₖ′ mₖ′ where c-suc-k: c (n + (Suc k)) = (tsₖ′,mₖ′,𝒮ₖ′)
      by (cases c (n + (Suc k)))

    obtain u-tsₖ u-sharedₖ u-mₖ where c′-k: c′ (n + k) = (u-tsₖ, u-mₖ, u-sharedₖ)
      by (cases c′ (n + k))

    from trace-preserves-length-ts [OF trace-k, of k 0] c-n c-k i-bound
    have i-bound-k: i < length tsₖ
      by simp

    from leq-l [rule-format, simplified l-k, of k] c-k c′-k
    have leq: length u-tsₖ = length tsₖ
      by simp

    note last-step = last-step [simplified c-k c-suc-k]
    from unchanged-suc-k c-suc-k
    have tsₖ′!i = ts!i
      by auto
    moreover from unchanged-k [rule-format, of k] c-k
    have unch-k-i: tsₖ!i=ts!i
      by auto
    ultimately have ts-eq: tsₖ!i=tsₖ′!i
      by simp

    from unchanged-i [simplified l-k, rule-format, OF - c-k c′-k]
    obtain
      u-ts-eq: u-tsₖ ! i = u-ts ! i and
```

unchanged-shared: ∀a∈u-owns. u-shared$_k$ a = u-shared a **and**
unchanged-shared-orig: ∀a∈u-owns. $\mathcal{S}_k$ a = $\mathcal{S}$ a **and**
unchanged-owns: ∀a∈u-owns. u-m$_k$ a = u-m a **and**
unchanged-owns-orig: ∀a∈u-owns. m$_k$ a = m a
by fastforce

from u-ts-eq u-ts-i
have u-ts$_k$-i: u-ts$_k$!i=(u-p,u-is,u-tmps,u-sb,u-dirty,u-owns,u-rels)
by auto
from sim [simplified l-k, rule-format, of k, OF - c-k c′-k]
obtain
ts-sim: (∀j<length ts$_k$. j ≠ i ⟶ u-ts$_k$! j = ts$_k$! j) **and**
shared-sim: (∀a. a ∉ u-owns ⟶ a ∉ owned (ts$_k$!i) ⟶ u-shared$_k$ a = $\mathcal{S}_k$ a) **and**
mem-sim: (∀a. a ∉ u-owns ⟶ u-m$_k$ a = m$_k$ a)
by (auto simp add: unch-k-i)

from unchanged-owns-orig unchanged-owns u-m-shared unchanged-shared
have unchanged-owns-shared: ∀a. a ∈ u-owns ⟶ a ∈ dom u-shared$_k$ ⟶ u-m$_k$ a = m$_k$ a
by (auto simp add: simp add: domIff)

from safe-l l-k safe-k
have safe-up-k: ∀x<k. safe-delayed (c′ (n + x))
apply clarsimp
done
from trace-preserves-simple-ownership-distinct [OF - trace-c′-l [simplified l-k] safe-up-k,
simplified c′-n, simplified, OF dist, of k] c′-k
have dist′: simple-ownership-distinct u-ts$_k$
by simp

from trace-preserves-simple-ownership-distinct [OF - trace-k, simplified c-n, simplified, OF dist-ts safe-orig, of k]
c-k
have dist-orig′: simple-ownership-distinct ts$_k$
by simp

from undo-local-step [OF last-step i-bound-k ts-eq safe-k [simplified c′-k] leq ts-sim u-ts$_k$-i mem-sim
unchanged-owns-shared shared-sim dist′ dist-orig′]
obtain u-ts′ u-shared′ u-m′ **where**
step′: (u-ts$_k$, u-m$_k$, u-shared$_k$) ⇒$_d$ (u-ts′, u-m′, u-shared′) **and**
ts-eq′: u-ts′ ! i = u-ts$_k$! i **and**
unchanged-shared′: (∀a∈u-owns. u-shared′ a = u-shared$_k$ a) **and**
unchanged-shared-orig′: (∀a∈u-owns. $\mathcal{S}_k$′ a = $\mathcal{S}_k$ a) **and**
unchanged-owns′: (∀a∈u-owns. u-m′ a = u-m$_k$ a) **and**
unchanged-owns-orig′: (∀a∈u-owns. m$_k$′ a = m$_k$ a) **and**

sim-ts′: (∀ j<length ts_k. j ≠ i ⟶ u-ts′ ! j = ts_k′ ! j) **and**
sim-shared′: (∀ a. a ∉ u-owns ⟶ a ∉ owned (ts_k ! i) ⟶ u-shared′ a = $\mathcal{S}_k$′ a) **and**
sim-m′: (∀ a. a ∉ u-owns ⟶ u-m′ a = m_k′ a)
by auto

define c″ **where** c″ == λl. if l ≤ n + k then c′ l else (u-ts′, u-m′, u-shared′)
have [simp]: ∀ x ≤ n + k. c″ x = c′ x
by (auto simp add: c″-def)
have [simp]: c″ (Suc (n + k)) = (u-ts′, u-m′, u-shared′)
by (auto simp add: c″-def)

from trace-c′-l l-k step′ c′-k **have** trace′: trace c″ n (Suc k)
apply (simp add: program-trace-def)
apply (clarsimp simp add: split-less-Suc)
done

from direct-computation.step-preserves-length-ts [OF last-step]
have leq-ts_k′: length ts_k′ = length ts_k.

with direct-computation.step-preserves-length-ts [OF step′] leq
have leq′: length u-ts′ = length ts_k
by simp
show ?thesis
apply (rule-tac x=c″ **in** exI)
apply (rule-tac x=Suc k **in** exI)
using safe-l l-k unchanged-i sim c-suc-k leq-l c′-n leq′
apply (clarsimp simp add: split-less-Suc split-le-Suc safe-k trace′ leq-ts_k′ sim-ts′ sim-shared′ sim-m′ unch-k-i

ts-eq′ u-ts-eq
unchanged-shared′ unchanged-shared unchanged-shared-orig unchanged-shared-orig′
unchanged-owns′ unchanged-owns
unchanged-owns-orig′ unchanged-owns-orig)
done
qed
qed
qed

locale program-safe-reach-upto = program +
fixes n **fixes** safe **fixes** c_0
assumes safe-config: ⟦k ≤ n; trace c 0 k; c 0 = c_0; l ≤ k ⟧ ⟹ safe (c l)

abbreviation (**in** program)
safe-reach-upto ≡ program-safe-reach-upto program-step

lemma (**in** program) safe-reach-upto-le:
assumes safe: safe-reach-upto n safe c_0
assumes m-n: m ≤ n

```
  shows safe-reach-upto m safe c₀
using safe m-n
apply (clarsimp simp add: program-safe-reach-upto-def)
  subgoal for k c
    apply (subgoal-tac k ≤ n)
     apply blast
    apply simp
  done
done


lemma (in program) last-action-of-thread:
assumes trace: trace c 0 k
shows
  — thread i never executes
  (∀l ≤ k. fst (c l)!i = fst (c k)!i)  ∨
  — thread i has a last step in the trace
  (∃last <  k.
    fst (c last)!i ≠ fst (c (Suc last))!i ∧
    (∀l. last < l ⟶ l ≤ k ⟶ fst (c l)!i = fst (c k)!i))
using trace
proof (induct k)
  case 0 thus ?case
    by auto
next
  case (Suc k)
  hence trace c 0 (Suc k) by simp
  then
  obtain
    trace-k: trace c 0 k and
    last-step: c k ⇒d c (Suc k)
    by (clarsimp simp add: program-trace-def)

  show ?case
  proof (cases fst (c k)!i = fst (c (Suc k))!i)
    case False
    then show ?thesis
      apply −
      apply (rule disjI2)
      apply (rule-tac x=k in exI)
      apply clarsimp
      apply (subgoal-tac l=Suc k)
      apply auto
      done
  next
    case True
    note idle-i = this


    {
```

assume same: (∀l≤k. fst (c l) ! i = fst (c k) ! i)
have ?thesis
apply –
apply (rule disjI1)
apply clarsimp
apply (case-tac l=Suc k)
apply (simp add: idle-i)
apply (rule same [simplified idle-i, rule-format])
apply simp
done
}
moreover
{
fix last
assume last-k: last < k
assume last-step: fst (c last) ! i ≠ fst (c (Suc last)) ! i
assume idle: (∀l>last. l ≤ k ⟶ fst (c l) ! i = fst (c k) ! i)
have ?thesis
apply –
apply (rule disjI2)
apply (rule-tac x=last **in** exI)
using last-k
apply (simp add: last-step)
using idle [simplified idle-i]
apply clarsimp
apply (case-tac l=Suc k)
apply clarsimp
apply clarsimp
done
}
moreover note Suc.hyps [OF trace-k]
ultimately
show ?thesis
by blast
qed
qed

lemma (**in** program) sequence-traces:
assumes trace1: trace c_1 0 k
assumes trace2: trace c_2 m l
assumes seq: c_2 m = c_1 k
assumes c-def: c = (λx. if x ≤ k then c_1 x else (c_2 (m + x −k)))
shows trace c 0 (k + l)
proof –
from trace1
interpret trace1: program-trace program-step c_1 0 k **.**
from trace2
interpret trace2: program-trace program-step c_2 m l **.**
{
fix x

```
    assume x-bound: x < (k + l)
    have c x ⇒d c (Suc x)
    proof (cases x < k)
      case True
      from trace1.step [OF True] True
      show ?thesis
        by (simp add: c-def)
    next
      case False
      hence k-x: k ≤ x
        by auto
      with x-bound have bound: x − k < l
        by auto
      from k-x have eq: (Suc (m + x) − k) = Suc (m + x − k)
        by simp
      from trace2.step [OF bound] k-x seq
      show ?thesis
        by (auto simp add: c-def eq)
    qed
  }
  thus ?thesis
    by (auto simp add: program-trace-def)
qed

theorem (in program-progress) safe-free-flowing-implies-safe-delayed:
  assumes init: initial c0
  assumes dist: simple-ownership-distinct (fst c0)
  assumes read-only-unowned: read-only-unowned (snd (snd c0)) (fst c0)
  assumes unowned-shared: unowned-shared (snd (snd c0)) (fst c0)
  assumes safe-reach-ff: safe-reach-upto n safe-free-flowing c0
  shows safe-reach-upto n safe-delayed c0
using safe-reach-ff
proof (induct n)
  case 0
  hence safe-reach-upto 0 safe-free-flowing c0 by simp
  hence safe-free-flowing c0
    by (auto simp add: program-safe-reach-upto-def)
  from initial-safe-free-flowing-implies-safe-delayed [OF init this]
  have safe-delayed c0.
  then show ?case
    by (simp add: program-safe-reach-upto-def)
next
  case (Suc n)
  hence safe-reach-suc: safe-reach-upto (Suc n) safe-free-flowing c0 by simp
   then interpret safe-reach-suc-inter: program-safe-reach-upto program-step (Suc n)
safe-free-flowing c0 .
  from safe-reach-upto-le [OF safe-reach-suc ]
  have safe-reach-n: safe-reach-upto n safe-free-flowing c0 by simp
  from Suc.hyps [OF this]
  have safe-delayed-reach-n: safe-reach-upto n safe-delayed c0.
```

```
  then interpret safe-delayed-reach-inter: program-safe-reach-upto program-step n
safe-delayed c0 .
 interpret direct-computation:
  computation direct-memop-step empty-storebuffer-step program-step λp p′ is sb. sb .
 show ?case
 proof (cases safe-reach-upto (Suc n) safe-delayed c0)
  case True thus ?thesis .
 next
  case False
  from safe-delayed-reach-n False
  obtain c where
   trace: trace c 0 (Suc n) and
   c-0: c 0 = c0 and
   safe-delayed-upto-n: ∀k≤n. safe-delayed (c k) and
   violation-delayed-suc: ¬ safe-delayed (c (Suc n))
  proof −
   from False
   obtain c k l where
    k-suc: k ≤ Suc n and
    trace-k: trace c 0 k and
    l-k: l ≤ k and
    violation: ¬ safe-delayed (c l) and
    start: c 0 = c0
    by (clarsimp simp add: program-safe-reach-upto-def)

   show ?thesis
   proof (cases k = Suc n)
    case False
    with k-suc have k ≤ n
     by auto
    from safe-delayed-reach-inter.safe-config [where c=c, OF this trace-k start l-k]
    have safe-delayed (c l).
    with violation have False by simp
    thus ?thesis ..
   next
    case True
    note k-suc-n = this
    from trace-k True have trace-n: trace c 0 n
     by (auto simp add: program-trace-def)
    show ?thesis
    proof (cases l=Suc n)
     case False
     with k-suc-n l-k have l ≤ n by simp
     from safe-delayed-reach-inter.safe-config [where c=c, OF - trace-n start this ]
     have safe-delayed (c l) by simp
     with violation have False by simp
     thus ?thesis ..
    next
     case True
     from safe-delayed-reach-inter.safe-config [where c=c, OF - trace-n start]
```

    have ∀k≤n. safe-delayed (c k) **by** simp
    with True k-suc-n trace-k start violation
    show ?thesis
      apply –
      apply (rule that)
      apply auto
      done
   qed
  qed
qed

from trace
interpret trace-inter: program-trace program-step c 0 Suc n **.**

from safe-reach-suc-inter.safe-config [**where** c=c, OF - trace c-0]
have safe-suc: safe-free-flowing (c (Suc n))
  by auto

obtain ts $\mathcal{S}$ m **where** c-suc: c (Suc n) = (ts,m,$\mathcal{S}$) **by** (cases c (Suc n))
from violation-delayed-suc c-suc
obtain i p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ **where**
  i-bound: i < length ts **and**
  ts-i: ts ! i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) **and**
  violation-i: ¬ map owned ts,map released ts,i ⊢(is,j,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{S}$)$\surd$
  by (fastforce simp add: safe-free-flowing-def safe-delayed-def)

from trace-preserves-unowned-shared [**where** c=c **and** n=0 **and** l=Suc n,
    simplified c-0, OF dist unowned-shared trace] safe-delayed-upto-n c-suc
have unowned-shared $\mathcal{S}$ ts **by** auto
then interpret unowned-shared $\mathcal{S}$ ts **.**

from violation-i **obtain** ins is′ **where** is: is = ins#is′
  by (cases is) (auto simp add: safe-delayed-direct-memop-state.Nil)
from safeE [OF safe-suc [simplified c-suc] i-bound ts-i]
have safe-i: map owned ts,i⊢(is, j, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{S}$)$\surd$**.**

define races **where** races == $\lambda\mathcal{R}$. (case ins of
   Read volatile a t ⇒ ($\mathcal{R}$ a = Some False) ∨ (¬ volatile ∧ a ∈ dom $\mathcal{R}$)
  | Write volatile a sop A L R W ⇒ (a ∈ dom $\mathcal{R}$ ∨ (volatile ∧ A ∩ dom $\mathcal{R}$ ≠ {}))
  | Ghost A L R W ⇒ (A ∩ dom $\mathcal{R}$ ≠ {})
  | RMW a t (D,f) cond ret A L R W ⇒ (if cond (j(t ↦ m a))
                    then a ∈ dom $\mathcal{R}$ ∨ A ∩ dom $\mathcal{R}$ ≠ {}
                    else $\mathcal{R}$ a = Some False)
  | - ⇒ False)

{
  assume no-race:

$\forall$ j. j < length ts $\longrightarrow$ j$\neq$i $\longrightarrow$ $\neg$ races (released (ts!j))
from safe-i
have map owned ts,map released ts,i $\vdash$(is,j,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{S}$)$\surd$
proof cases
  case Read
  thus ?thesis
    using is no-race
    by (auto simp add: races-def intro: safe-delayed-direct-memop-state.intros)
next
  case WriteNonVolatile
  thus ?thesis
    using is no-race
    by (auto simp add: races-def intro: safe-delayed-direct-memop-state.intros)
next
  case WriteVolatile
  thus ?thesis
    using is no-race
    apply (clarsimp simp add: races-def)
    apply (rule safe-delayed-direct-memop-state.intros)
    apply auto
    done
next
  case Fence
  thus ?thesis
    using is no-race
    by (auto simp add: races-def intro: safe-delayed-direct-memop-state.intros)
next
  case Ghost
  thus ?thesis
    using is no-race
    apply (clarsimp simp add: races-def)
    apply (rule safe-delayed-direct-memop-state.intros)
    apply auto
    done
next
  case RMWReadOnly
  thus ?thesis
    using is no-race
    by (auto simp add: races-def intro: safe-delayed-direct-memop-state.intros)
next
  case (RMWWrite cond t a - - A - $\mathcal{O}$)
  thus ?thesis
    using is no-race unowned-shared′ [rule-format, of a] ts-i
    apply (clarsimp simp add: races-def)
    apply (rule safe-delayed-direct-memop-state.RMWWrite)
    apply auto
    apply force
    done
next
  case Nil **with** is **show** ?thesis **by** auto

```
  qed
}
with violation-i
obtain j where
  j-bound: j < length ts and
  neq-j-i: j ≠ i and
  race: races (released (ts!j))
  by auto

obtain pj isj jj sbj 𝒟j 𝒪j ℛj where
  ts-j: ts!j = (pj,isj,jj,sbj,𝒟j,𝒪j,ℛj)
  apply (cases ts!j)
  apply force
  done

from race
have ℛj-non-empty: ℛj ≠ Map.empty
  by (auto simp add: ts-j races-def split: instr.splits if-split-asm)

{
  assume idle-j: ∀l≤Suc n. fst (c l) ! j = fst (c (Suc n)) ! j
  have ?thesis
  proof –
    from idle-j [rule-format, of 0] c-suc  c-0 ts-j
    have c0-j: fst c0 ! j = ts!j
      by clarsimp
    from trace-preserves-length-ts [OF trace, of 0 Suc n] c-0 c-suc
    have length (fst c0) = length ts
      by clarsimp
    with j-bound have j < length (fst c0)
      by simp
    with nth-mem [OF this] init c0-j ts-j
    have ℛj = Map.empty
      by (auto simp add: initial-def)
    with ℛj-non-empty have False
      by simp
    thus ?thesis ..
  qed
}
moreover
{
  fix last
  assume last-bound: last<Suc n
  assume last-step-changed-j: fst (c last) ! j ≠ fst (c (Suc last)) ! j
  assume idle-rest: ∀l>last. l ≤ Suc n ⟶ fst (c l) ! j = fst (c (Suc n)) ! j
  have ?thesis
  proof –
    obtain tsl 𝒮l ml where
       c-last: c last = (tsl,ml,𝒮l)
      by (cases c last)
```

obtain ts_l' $\mathcal{S}_l'$ m_l' **where**
c-last′: c (Suc last) = (ts_l',m_l',$\mathcal{S}_l'$)
by (cases c (Suc last))
from idle-rest [rule-format, of Suc last] c-suc c-last′ last-bound
have ts_l'-j: ts_l'!j = ts!j
by auto

from last-step-changed-j c-last c-last′
have j-changed: ts_l!j ≠ ts_l'!j
by auto

from trace-inter.step [OF last-bound] c-last c-last′
have last-step: (ts_l,m_l,$\mathcal{S}_l$) $\Rightarrow_d$ (ts_l',m_l',$\mathcal{S}_l'$)
by simp

obtain p_l is_l j_l sb_l $\mathcal{D}_l$ $\mathcal{O}_l$ $\mathcal{R}_l$ **where**
ts_l-j: ts_l!j = (p_l,is_l,j_l,sb_l,$\mathcal{D}_l$,$\mathcal{O}_l$,$\mathcal{R}_l$)
apply (cases ts_l!j)
apply force
done

from trace-preserves-length-ts [OF trace, of last Suc n] c-last c-suc last-bound
have leq_l: length ts_l = length ts
by simp
with j-bound **have** j-$bound_l$: j < length ts_l
by simp

from trace **have** trace-n: trace c 0 n
by (auto simp add: program-trace-def)

from safe-delayed-reach-inter.safe-config [**where** k=n **and** c=c **and** l=last, OF - trace-n c-0] last-bound c-last
have safe-delayed-last: safe-delayed (ts_l,m_l,$\mathcal{S}_l$)
by auto

from safe-delayed-reach-inter.safe-config [**where** c=c, OF - trace-n c-0]
have safe-delayed-upto-n: ∀x<n. safe-delayed (c (0 + x))
by auto
from trace-preserves-simple-ownership-distinct [**where** c=c **and** n=0 **and** l=last, simplified c-0, OF dist trace-n safe-delayed-upto-n]
last-bound c-last
have dist-last: simple-ownership-distinct ts_l
by auto

from trace-preserves-read-only-unowned [**where** c=c **and** n=0 **and** l=last, simplified c-0, OF dist read-only-unowned trace-n safe-delayed-upto-n]
last-bound c-last
have ro-last-last: read-only-unowned $\mathcal{S}_l$ ts_l
by auto

```
from safe-delayed-reach-inter.safe-config [where c=c, OF - trace-n c-0]
have safe-delayed-upto-suc-n: ∀x<Suc n. safe-delayed (c (0 + x))
  by auto

 from trace-preserves-simple-ownership-distinct [where c=c and n=0 and l=Suc
last,
  simplified c-0, OF dist trace safe-delayed-upto-suc-n]
  last-bound c-last′
have dist-last′: simple-ownership-distinct tsₗ′
  by auto
from trace last-bound have trace-last: trace c 0 last
  by (auto simp add: program-trace-def)

from trace last-bound have trace-rest: trace c (Suc last) (n − last)
  by (auto simp add: program-trace-def)

from idle-rest last-bound
have idle-rest′:
  ∀l≤n − last.
        ∀tsₗ 𝒮ₗ mₗ. c (Suc last + l) = (tsₗ, mₗ, 𝒮ₗ) ⟶ tsₗ ! j = tsₗ′ ! j
  apply clarsimp
  apply (drule-tac x=Suc (last + l) in spec)
  apply (auto simp add: c-last′ c-suc tsₗ′-j)
  done

from safe-delayed-upto-suc-n [rule-format, of last] last-bound
have safe-delayed-last: safe-delayed (tsₗ, mₗ, 𝒮ₗ)
  by (auto simp add: c-last)
from safe-delayedE [OF this j-boundₗ tsₗ-j]
have safeₗ: map owned tsₗ,map released tsₗ,j⊢(isₗ, jₗ, mₗ, 𝒟ₗ, 𝒪ₗ, 𝒮ₗ)√.

from safe-delayed-reach-inter.safe-config [where c=c, OF - trace-n c-0]
have safe-delayed-upto-last: ∀x<n − last. safe-delayed (c (Suc (last + x)))
  by auto

from last-step
show ?thesis
proof (cases)
  case (Program i′ - - - - - - - p′ is′)
  with j-changed j-boundₗ tsₗ-j
  obtain
    tsₗ′: tsₗ′ = tsₗ[j:=(p′,isₗ@is′,jₗ,sbₗ,𝒟ₗ,𝒪ₗ,ℛₗ)] and
    𝒮ₗ′: 𝒮ₗ′=𝒮ₗ and
    mₗ′: mₗ′=mₗ and
    prog-step: jₗ⊢ pₗ →ₚ (p′, is′)
    by (cases i′=j) auto
  from tsₗ′-j tsₗ′ ts-j j-boundₗ
  obtain eqs: p′=pⱼ isₗ@is′=isⱼ jₗ=jⱼ 𝒟ₗ=𝒟ⱼ 𝒪ₗ=𝒪ⱼ ℛₗ=ℛⱼ
```

by auto

from undo-local-steps [**where** c=c, OF trace-rest c-last′ idle-rest′ safe-delayed-last, simplified ts_l′,
simplified,
OF j-bound$_l$ ts_l-j [simplified], simplified m_l′ $\mathcal{S}_l$′, simplified, OF dist-last
dist-last′ [simplified ts_l′,simplified] safe-delayed-upto-last]
obtain c′ k **where**
k-bound: k ≤ n − last **and**
trace-c′: trace c′ (Suc last) k **and**
c′-first: c′ (Suc last) = (ts_l, m_l, $\mathcal{S}_l$) **and**
c′-leq: (∀x≤k. length (fst (c′ (Suc (last + x)))) = length (fst (c (Suc (last + x))))) **and**
c′-safe: (∀x<k. safe-delayed (c′ (Suc (last + x)))) **and**
c′-unsafe: (k < n − last ⟶ ¬ safe-delayed (c′ (Suc (last + k)))) **and**
c′-unch:
(∀x≤k. ∀ts_x $\mathcal{S}_x$ m_x.
c (Suc (last + x)) = (ts_x, m_x, $\mathcal{S}_x$) ⟶
(∀ts_x′ $\mathcal{S}_x$′ m_x′.
c′ (Suc (last + x)) = (ts_x′, m_x′, $\mathcal{S}_x$′) ⟶
ts_x′ ! j = ts_l ! j ∧
(∀a∈$\mathcal{O}_l$. $\mathcal{S}_x$′ a = $\mathcal{S}_l$ a) ∧
(∀a∈$\mathcal{O}_l$. $\mathcal{S}_x$ a = $\mathcal{S}_l$ a) ∧
(∀a∈$\mathcal{O}_l$. m_x′ a = m_l a) ∧ (∀a∈$\mathcal{O}_l$. m_x a = m_l a))) **and**
c′-sim:
(∀x≤k. ∀ts_x $\mathcal{S}_x$ m_x.
c (Suc (last + x)) = (ts_x, m_x, $\mathcal{S}_x$) ⟶
(∀ts_x′ $\mathcal{S}_x$′ m_x′.
c′ (Suc (last + x)) = (ts_x′, m_x′, $\mathcal{S}_x$′) ⟶
(∀ja<length ts_x. ja ≠ j ⟶ ts_x′ ! ja = ts_x ! ja) ∧
(∀a. a ∉ $\mathcal{O}_l$ ⟶ $\mathcal{S}_x$′ a = $\mathcal{S}_x$ a) ∧
(∀a. a ∉ $\mathcal{O}_l$ ⟶ m_x′ a = m_x a)))
by auto

obtain c-undo **where** c-undo: c-undo = (λx. if x ≤ last then c x else c′ (Suc last + x − last))
by blast
have c-undo-0: c-undo 0 = c_0
by (auto simp add: c-undo c-0)
from sequence-traces [OF trace-last trace-c′, simplified c-last, OF c′-first c-undo]
have trace-undo: trace c-undo 0 (last + k) **.**
obtain u-ts u-shared u-m **where**
c-undo-n: c-undo n = (u-ts,u-m, u-shared)
by (cases c-undo n)
with last-bound c′-first c-last
have c′-suc: c′ (Suc n) = (u-ts,u-m, u-shared)
apply (auto simp add: c-undo split: if-split-asm)
apply (subgoal-tac n=last)
apply auto

```
  done


show ?thesis
proof (cases k < n − last)
  case True
  with c′-unsafe have unsafe: ¬ safe-delayed (c-undo (last + k))
    by (auto simp add: c-undo c-last c′-first)
  from True have last + k ≤ n
    by auto
  from safe-delayed-reach-inter.safe-config [OF this trace-undo, of last + k]
  have safe-delayed (c-undo (last + k))
    by (auto simp add: c-undo c-0)
  with unsafe have False by simp
  thus ?thesis ..
next
  case False
  with k-bound have k: k = n − last
    by auto
  have eq′: Suc (last + (n − last)) = Suc n
    using last-bound
    by simp
  from c′-unch [rule-format, of k, simplified k eq′, OF - c-suc c′-suc]
  obtain u-ts-j: u-ts!j = tsₗ!j and
    shared-unch: ∀a∈𝒪ₗ. u-shared a = 𝒮ₗ a and
    shared-orig-unch: ∀a∈𝒪ₗ. 𝒮 a = 𝒮ₗ a and
    mem-unch: ∀a∈𝒪ₗ. u-m a = mₗ a and
    mem-unch-orig: ∀a∈𝒪ₗ. m a = mₗ a
    by auto

  from c′-sim [rule-format, of k, simplified k eq′, OF - c-suc c′-suc] i-bound neq-j-i
  obtain u-ts-i: u-ts!i = ts!i and
     shared-sim: ∀a. a ∉ 𝒪ₗ ⟶ u-shared a = 𝒮 a and
     mem-sim: ∀a. a ∉ 𝒪ₗ ⟶ u-m a = m a
    by auto

  from c′-leq [rule-format, of k] c′-suc c-suc
  have leq-u-ts: length u-ts = length ts
    by (auto simp add: eq′ k)

  from j-bound leq-u-ts
  have j-bound-u: j < length u-ts
    by simp
  from i-bound leq-u-ts
  have i-bound-u: i < length u-ts
    by simp
  from k last-bound have l-k-eq: last + k = n
    by auto
  from safe-delayed-reach-inter.safe-config [OF - trace-undo, simplified l-k-eq]
    k c-0 last-bound
```

```
have safe-delayed-c-undo′: ∀x≤ n. safe-delayed (c-undo x)
  by (auto simp add: c-undo split: if-split-asm)
hence safe-delayed-c-undo: ∀x<n. safe-delayed (c-undo x)
  by (auto)
from trace-preserves-simple-ownership-distinct [OF - trace-undo,
  simplified l-k-eq c-undo-0, simplified, OF dist this, of n] dist c-undo-n
have dist-u-ts: simple-ownership-distinct u-ts
  by auto
then interpret dist-u-ts-inter: simple-ownership-distinct u-ts .

{
  fix a
  have u-m a = m a
  proof (cases a ∈ 𝒪ₗ)
    case True with mem-unch
    have u-m a = mₗ a
      by auto
    moreover
    from True mem-unch-orig
    have m a = mₗ a
      by auto
    ultimately show ?thesis by simp
  next
    case False
    with mem-sim
    show ?thesis
      by auto
  qed
} hence u-m-eq: u-m = m by − (rule ext, auto)

{
  fix a
  have u-shared a = 𝒮 a
  proof (cases a ∈ 𝒪ₗ)
    case True with shared-unch
    have u-shared a = 𝒮ₗ a
      by auto
    moreover
    from True shared-orig-unch
    have 𝒮 a = 𝒮ₗ a
      by auto
    ultimately show ?thesis by simp
  next
    case False
    with shared-sim
    show ?thesis
      by auto
  qed
} hence u-shared-eq: u-shared = 𝒮 by − (rule ext, auto)
```

{
assume safe: map owned u-ts,map released u-ts,i ⊢(is,j,u-m,$\mathcal{D}$,$\mathcal{O}$,u-shared)$\surd$
then have False
proof cases
case Read
then show ?thesis
using ts-i ts$_{\mathsf{l}}$-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
by (auto simp add:eqs races-def split: if-split-asm)
next
case WriteNonVolatile
then show ?thesis
using ts-i ts$_{\mathsf{l}}$-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
by (auto simp add:eqs races-def split: if-split-asm)
next
case WriteVolatile
then show ?thesis
using ts-i ts$_{\mathsf{l}}$-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
apply (auto simp add:eqs races-def split: if-split-asm)
apply fastforce
done
next
case Fence
then show ?thesis
using ts-i ts$_{\mathsf{l}}$-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
by (auto simp add:eqs races-def split: if-split-asm)
next
case Ghost
then show ?thesis
using ts-i ts$_{\mathsf{l}}$-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
apply (auto simp add:eqs races-def split: if-split-asm)
apply fastforce
done
next
case (RMWReadOnly cond t a D f ret A L R W)
then show ?thesis
using ts-i ts$_{\mathsf{l}}$-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
by (auto simp add:eqs races-def u-shared-eq u-m-eq split: if-split-asm)
next
case RMWWrite
then show ?thesis
using ts-i ts$_{\mathsf{l}}$-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
apply (auto simp add:eqs races-def u-shared-eq u-m-eq split: if-split-asm)
apply fastforce+
done
next
case Nil
then show ?thesis
using ts-i ts$_{\mathsf{l}}$-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
by (auto simp add:eqs races-def split: if-split-asm)
qed

}
hence ¬ safe-delayed (u-ts, u-m, u-shared)
apply (clarsimp simp add: safe-delayed-def)
apply (rule-tac x=i **in** exI)
using u-ts-i ts-i i-bound-u
apply auto
done
moreover
from safe-delayed-c-undo′ [rule-format, of n] c-undo-n
have safe-delayed (u-ts, u-m, u-shared)
by simp
ultimately have False
by simp
thus ?thesis
by simp
qed
next
case (Memop i′ - - - - - - - is_l′ j_l′ sb_l′ $\mathcal{D}_l$′ $\mathcal{O}_l$′ $\mathcal{R}_l$′)
with j-changed j-bound$_l$ ts$_l$-j
obtain
ts$_l$′: $\mathrm{ts}_l' = \mathrm{ts}_l[j:=(p_l,\mathrm{is}_l',j_l',\mathrm{sb}_l',\mathcal{D}_l',\mathcal{O}_l',\mathcal{R}_l')]$ **and**
mem-step: $(\mathrm{is}_l, j_l, \mathrm{sb}_l, m_l, \mathcal{D}_l, \mathcal{O}_l, \mathcal{R}_l,\mathcal{S}_l) \rightarrow$
$(\mathrm{is}_l', j_l', \mathrm{sb}_l', m_l', \mathcal{D}_l', \mathcal{O}_l', \mathcal{R}_l', \mathcal{S}_l')$
by (cases i′=j) auto

from mem-step
show ?thesis
proof (cases)
case (Read volatile a t)
then obtain
is$_l$: is_l = Read volatile a t # is_l′ **and**
j_l′: $j_l' = j_l(t \mapsto m_l\ a)$ **and**
sb$_l$′: $\mathrm{sb}_l'=\mathrm{sb}_l$ **and**
$\mathcal{D}_l$′: $\mathcal{D}_l'=\mathcal{D}_l$ **and**
$\mathcal{O}_l$′: $\mathcal{O}_l' = \mathcal{O}_l$ **and**
$\mathcal{R}_l$′: $\mathcal{R}_l'= \mathcal{R}_l$ **and**
$\mathcal{S}_l$′: $\mathcal{S}_l'=\mathcal{S}_l$ **and**
m$_l$′: $m_l' = m_l$
by auto
note eqs′ = j_l′ sb$_l$′ $\mathcal{D}_l$′ $\mathcal{O}_l$′ $\mathcal{R}_l$′ $\mathcal{S}_l$′ m$_l$′
from ts$_l$′-j ts$_l$′ ts-j j-bound$_l$ eqs′
obtain eqs: $p_l=p_j$ $\mathrm{is}_l'=\mathrm{is}_j$ $j_l(t \mapsto m_l\ a)=j_j$ $\mathcal{D}_l=\mathcal{D}_j$ $\mathcal{O}_l=\mathcal{O}_j$ $\mathcal{R}_l=\mathcal{R}_j$
by auto

from undo-local-steps [**where** c=c, OF trace-rest c-last′ idle-rest′ safe-delayed-last, simplified ts$_l$′,
simplified,
OF j-bound$_l$ ts$_l$-j [simplified], simplified m$_l$′ $\mathcal{S}_l$′, simplified, OF dist-last
dist-last′ [simplified ts$_l$′,simplified] safe-delayed-upto-last]

obtain c′ k **where**
k-bound: k ≤ n − last **and**
trace-c′: trace c′ (Suc last) k **and**
c′-first: c′ (Suc last) = (ts_l, m_l, $\mathcal{S}_l$) **and**
c′-leq: (∀x≤k. length (fst (c′ (Suc (last + x)))) = length (fst (c (Suc (last + x))))) **and**
c′-safe: (∀x<k. safe-delayed (c′ (Suc (last + x)))) **and**
c′-unsafe: (k < n − last ⟶ ¬ safe-delayed (c′ (Suc (last + k)))) **and**
c′-unch:
(∀x≤k. ∀ts_x $\mathcal{S}_x$ m_x.
c (Suc (last + x)) = (ts_x, m_x, $\mathcal{S}_x$) ⟶
(∀ts_x′ $\mathcal{S}_x$′ m_x′.
c′ (Suc (last + x)) = (ts_x′, m_x′, $\mathcal{S}_x$′) ⟶
ts_x′ ! j = ts_l ! j ∧
(∀a∈$\mathcal{O}_l$. $\mathcal{S}_x$′ a = $\mathcal{S}_l$ a) ∧
(∀a∈$\mathcal{O}_l$. $\mathcal{S}_x$ a = $\mathcal{S}_l$ a) ∧
(∀a∈$\mathcal{O}_l$. m_x′ a = m_l a) ∧ (∀a∈$\mathcal{O}_l$. m_x a = m_l a))) **and**
c′-sim:
(∀x≤k. ∀ts_x $\mathcal{S}_x$ m_x.
c (Suc (last + x)) = (ts_x, m_x, $\mathcal{S}_x$) ⟶
(∀ts_x′ $\mathcal{S}_x$′ m_x′.
c′ (Suc (last + x)) = (ts_x′, m_x′, $\mathcal{S}_x$′) ⟶
(∀ja<length ts_x. ja ≠ j ⟶ ts_x′ ! ja = ts_x ! ja) ∧
(∀a. a ∉ $\mathcal{O}_l$ ⟶ $\mathcal{S}_x$′ a = $\mathcal{S}_x$ a) ∧
(∀a. a ∉ $\mathcal{O}_l$ ⟶ m_x′ a = m_x a)))
by (clarsimp simp add: $\mathcal{O}_l$′)
obtain c-undo **where** c-undo: c-undo = (λx. if x ≤ last then c x else c′ (Suc last + x − last))
by blast
have c-undo-0: c-undo 0 = c_0
by (auto simp add: c-undo c-0)
from sequence-traces [OF trace-last trace-c′, simplified c-last, OF c′-first c-undo]
have trace-undo: trace c-undo 0 (last + k) **.**
obtain u-ts u-shared u-m **where**
c-undo-n: c-undo n = (u-ts,u-m, u-shared)
by (cases c-undo n)
with last-bound c′-first c-last
have c′-suc: c′ (Suc n) = (u-ts,u-m, u-shared)
apply (auto simp add: c-undo split: if-split-asm)
apply (subgoal-tac n=last)
apply auto
done

show ?thesis
proof (cases k < n − last)
case True
with c′-unsafe **have** unsafe: ¬ safe-delayed (c-undo (last + k))
by (auto simp add: c-undo c-last c′-first)
from True **have** last + k ≤ n

by auto
from safe-delayed-reach-inter.safe-config [OF this trace-undo, of last + k]
have safe-delayed (c-undo (last + k))
by (auto simp add: c-undo c-0)
with unsafe **have** False **by** simp
thus ?thesis **..**
next
case False
with k-bound **have** k: k = n − last
by auto
have eq′: Suc (last + (n − last)) = Suc n
using last-bound
by simp
from c′-unch [rule-format, of k, simplified k eq′, OF - c-suc c′-suc]
obtain u-ts-j: u-ts!j = ts_l!j **and**
shared-unch: $\forall a \in \mathcal{O}_l$. u-shared a = $\mathcal{S}_l$ a **and**
shared-orig-unch: $\forall a \in \mathcal{O}_l$. $\mathcal{S}$ a = $\mathcal{S}_l$ a **and**
mem-unch: $\forall a \in \mathcal{O}_l$. u-m a = m_l a **and**
mem-unch-orig: $\forall a \in \mathcal{O}_l$. m a = m_l a
by auto

from c′-sim [rule-format, of k, simplified k eq′, OF - c-suc c′-suc] i-bound neq-j-i
obtain u-ts-i: u-ts!i = ts!i **and**
shared-sim: $\forall$a. a $\notin \mathcal{O}_l \longrightarrow$ u-shared a = $\mathcal{S}$ a **and**
mem-sim: $\forall$a. a $\notin \mathcal{O}_l \longrightarrow$ u-m a = m a
by auto

from c′-leq [rule-format, of k] c′-suc c-suc
have leq-u-ts: length u-ts = length ts
by (auto simp add: eq′ k)

from j-bound leq-u-ts
have j-bound-u: j < length u-ts
by simp
from i-bound leq-u-ts
have i-bound-u: i < length u-ts
by simp
from k last-bound **have** l-k-eq: last + k = n
by auto
from safe-delayed-reach-inter.safe-config [OF - trace-undo, simplified l-k-eq]
k c-0 last-bound
have safe-delayed-c-undo′: $\forall x \le n$. safe-delayed (c-undo x)
by (auto simp add: c-undo split: if-split-asm)
hence safe-delayed-c-undo: $\forall x < n$. safe-delayed (c-undo x)
by (auto)
from trace-preserves-simple-ownership-distinct [OF - trace-undo,
simplified l-k-eq c-undo-0, simplified, OF dist this, of n] dist c-undo-n
have dist-u-ts: simple-ownership-distinct u-ts
by auto
then interpret dist-u-ts-inter: simple-ownership-distinct u-ts **.**

```
{
 fix a
 have u-m a = m a
 proof (cases a ∈ 𝒪ₗ)
   case True with mem-unch
   have u-m a = mₗ a
     by auto
   moreover
   from True mem-unch-orig
   have m a = mₗ a
     by auto
   ultimately show ?thesis by simp
 next
   case False
   with mem-sim
   show ?thesis
     by auto
 qed
} hence u-m-eq: u-m = m by − (rule ext, auto)

{
 fix a
 have u-shared a = 𝒮 a
 proof (cases a ∈ 𝒪ₗ)
   case True with shared-unch
   have u-shared a = 𝒮ₗ a
     by auto
   moreover
   from True shared-orig-unch
   have 𝒮 a = 𝒮ₗ a
     by auto
   ultimately show ?thesis by simp
 next
   case False
   with shared-sim
   show ?thesis
     by auto
 qed
} hence u-shared-eq: u-shared = 𝒮 by − (rule ext, auto)

{
assume safe: map owned u-ts,map released u-ts,i ⊢(is,j,u-m,𝒟,𝒪,u-shared)√
 then have False
 proof cases
   case Read
   then show ?thesis
   using  ts-i tsₗ-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
     by (auto simp add:eqs races-def split: if-split-asm)
```

```
  next
    case WriteNonVolatile
    then show ?thesis
    using  ts-i ts_l-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
      by (auto simp add:eqs races-def split: if-split-asm)
  next
    case WriteVolatile
    then show ?thesis
    using  ts-i ts_l-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
      apply (auto simp add:eqs races-def split: if-split-asm)
      apply fastforce
      done
  next
    case Fence
    then show ?thesis
    using  ts-i ts_l-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
      by (auto simp add:eqs races-def split: if-split-asm)
  next
    case Ghost
    then show ?thesis
    using  ts-i ts_l-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
      apply (auto simp add:eqs races-def split: if-split-asm)
      apply fastforce
      done
  next
    case (RMWReadOnly cond t a D f ret A L R W)
    then show ?thesis
    using  ts-i ts_l-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
      by (auto simp add:eqs races-def u-shared-eq u-m-eq split: if-split-asm)
  next
    case RMWWrite
    then show ?thesis
    using  ts-i ts_l-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
      apply (auto simp add:eqs races-def u-shared-eq u-m-eq split: if-split-asm)
      apply fastforce+
      done
  next
    case Nil
    then show ?thesis
    using  ts-i ts_l-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
      by (auto simp add:eqs races-def split: if-split-asm)
  qed
}
hence ¬ safe-delayed (u-ts, u-m, u-shared)
  apply (clarsimp simp add: safe-delayed-def)
  apply (rule-tac x=i in exI)
  using u-ts-i ts-i i-bound-u
  apply auto
  done
moreover
```

from safe-delayed-c-undo′ [rule-format, of n] c-undo-n
have safe-delayed (u-ts, u-m, u-shared)
by simp
ultimately have False
by simp
thus ?thesis
by simp
qed
next
case (WriteNonVolatile a D f A L R W)
then obtain
is_l: is_l = Write False a (D, f) A L R W # is_l′ **and**
j_l′: j_l′ = j_l **and**
sb_l′: sb_l′=sb_l **and**
$\mathcal{D}_l$′: $\mathcal{D}_l$′=$\mathcal{D}_l$ **and**
$\mathcal{O}_l$′: $\mathcal{O}_l$′ = $\mathcal{O}_l$ **and**
$\mathcal{R}_l$′: $\mathcal{R}_l$′= $\mathcal{R}_l$ **and**
$\mathcal{S}_l$′: $\mathcal{S}_l$′=$\mathcal{S}_l$ **and**
m_l′: m_l′ = m_l(a:=f j_l)
by auto
note eqs′ = j_l′ sb_l′ $\mathcal{D}_l$′ $\mathcal{O}_l$′ $\mathcal{R}_l$′ $\mathcal{S}_l$′ m_l′
from ts_l′-j ts_l′ ts-j j-bound$_l$ eqs′
obtain eqs: p_l=p_j is_l′=is_j j_l=j_j $\mathcal{D}_l$=$\mathcal{D}_j$ $\mathcal{O}_l$=$\mathcal{O}_j$
$\mathcal{R}_l$=$\mathcal{R}_j$
by auto

from safe$_l$ [simplified is_l]
obtain a-owned: $a \in \mathcal{O}_l$ **and** a-unshared: $a \notin$ dom $\mathcal{S}_l$
by cases auto
have m_l-unch-unowned: $\forall a'.\ a' \notin \mathcal{O}_l \longrightarrow m_l\ a' = (m_l(a := f\ j_l))\ a'$
using a-owned **by** auto

have m_l-unch-unshared: $\forall a'.\ a' \in \mathcal{O}_l \longrightarrow a' \in \text{dom}\ \mathcal{S}_l \longrightarrow m_l\ a' = (m_l(a := f\ j_l))$
a'
using a-unshared **by** auto

from undo-local-steps [**where** c=c, OF trace-rest c-last′ idle-rest′ safe-delayed-last,
simplified ts_l′,
simplified,
OF j-bound$_l$ ts_l-j [simplified], simplified m_l′ $\mathcal{S}_l$′,OF m_l-unch-unowned
m_l-unch-unshared, simplified,
OF dist-last dist-last′ [simplified ts_l′,simplified] safe-delayed-upto-last]

obtain c′ k **where**
k-bound: $k \leq n - \text{last}$ **and**
trace-c′: trace c′ (Suc last) k **and**
c′-first: c′ (Suc last) = (ts_l, m_l, $\mathcal{S}_l$) **and**
c′-leq: ($\forall$x≤k. length (fst (c′ (Suc (last + x)))) = length (fst (c (Suc (last +
x))))) **and**
c′-safe: ($\forall$x<k. safe-delayed (c′ (Suc (last + x)))) **and**

```
      c'-unsafe: (k < n − last ⟶ ¬ safe-delayed (c' (Suc (last + k)))) and
      c'-unch:
       (∀x≤k. ∀tsₓ 𝒮ₓ mₓ.
          c (Suc (last + x)) = (tsₓ, mₓ, 𝒮ₓ) ⟶
          (∀tsₓ' 𝒮ₓ' mₓ'.
             c' (Suc (last + x)) = (tsₓ', mₓ', 𝒮ₓ') ⟶
             tsₓ' ! j = tsₗ ! j ∧
             (∀a∈𝒪ₗ. 𝒮ₓ' a = 𝒮ₗ a) ∧
             (∀a∈𝒪ₗ. 𝒮ₓ a = 𝒮ₗ a) ∧
             (∀a∈𝒪ₗ. mₓ' a = mₗ a) ∧ (∀a'∈𝒪ₗ. mₓ a' = (mₗ(a := f jₗ)) a'))) and
      c'-sim:
       (∀x≤k. ∀tsₓ 𝒮ₓ mₓ.
          c (Suc (last + x)) = (tsₓ, mₓ, 𝒮ₓ) ⟶
          (∀tsₓ' 𝒮ₓ' mₓ'.
             c' (Suc (last + x)) = (tsₓ', mₓ', 𝒮ₓ') ⟶
             (∀ja<length tsₓ. ja ≠ j ⟶ tsₓ' ! ja = tsₓ ! ja) ∧
             (∀a. a ∉ 𝒪ₗ ⟶ 𝒮ₓ' a = 𝒮ₓ a) ∧
             (∀a. a ∉ 𝒪ₗ ⟶ mₓ' a = mₓ a)))
      by (clarsimp simp add: 𝒪ₗ')

     obtain c-undo where c-undo:  c-undo = (λx. if x ≤ last then c x else c' (Suc
last + x − last))
      by blast
    have c-undo-0: c-undo 0 = c₀
      by (auto simp add: c-undo c-0)
   from sequence-traces [OF trace-last trace-c', simplified c-last, OF c'-first c-undo]
    have trace-undo: trace c-undo 0 (last + k) .
    obtain u-ts u-shared u-m where
      c-undo-n: c-undo n = (u-ts,u-m, u-shared)
      by (cases c-undo n)
    with last-bound c'-first c-last
    have c'-suc: c' (Suc n) = (u-ts,u-m, u-shared)
      apply (auto simp add: c-undo split: if-split-asm)
      apply (subgoal-tac n=last)
      apply auto
      done


    show ?thesis
    proof (cases k < n − last)
      case True
      with c'-unsafe have unsafe: ¬ safe-delayed (c-undo (last + k))
        by (auto simp add: c-undo c-last c'-first)
      from True have last + k ≤ n
        by auto
      from safe-delayed-reach-inter.safe-config [OF this trace-undo, of last + k]
      have safe-delayed (c-undo (last + k))
        by (auto simp add: c-undo c-0)
      with unsafe have False by simp
      thus ?thesis ..
```

next
case False
with k-bound **have** k: k = n − last
by auto
have eq′: Suc (last + (n − last)) = Suc n
using last-bound
by simp
from c′-unch [rule-format, of k, simplified k eq′, OF - c-suc c′-suc]
obtain u-ts-j: u-ts!j = ts_l!j **and**
shared-unch: ∀a∈$\mathcal{O}_l$. u-shared a = $\mathcal{S}_l$ a **and**
shared-orig-unch: ∀a∈$\mathcal{O}_l$. $\mathcal{S}$ a = $\mathcal{S}_l$ a **and**
mem-unch: ∀a∈$\mathcal{O}_l$. u-m a = m_l a **and**
mem-unch-orig: ∀a′∈$\mathcal{O}_l$. m a′ = (m_l(a := f j_l)) a′
by auto

from c′-sim [rule-format, of k, simplified k eq′, OF - c-suc c′-suc] i-bound neq-j-i
obtain u-ts-i: u-ts!i = ts!i **and**
shared-sim: ∀a. a ∉ $\mathcal{O}_l$ ⟶ u-shared a = $\mathcal{S}$ a **and**
mem-sim: ∀a. a ∉ $\mathcal{O}_l$ ⟶ u-m a = m a
by auto

from c′-leq [rule-format, of k] c′-suc c-suc
have leq-u-ts: length u-ts = length ts
by (auto simp add: eq′ k)

from j-bound leq-u-ts
have j-bound-u: j < length u-ts
by simp
from i-bound leq-u-ts
have i-bound-u: i < length u-ts
by simp
from k last-bound **have** l-k-eq: last + k = n
by auto
from safe-delayed-reach-inter.safe-config [OF - trace-undo, simplified l-k-eq]
k c-0 last-bound
have safe-delayed-c-undo′: ∀x≤n. safe-delayed (c-undo x)
by (auto simp add: c-undo split: if-split-asm)
hence safe-delayed-c-undo: ∀x<n. safe-delayed (c-undo x)
by auto
from trace-preserves-simple-ownership-distinct [OF - trace-undo,
simplified l-k-eq c-undo-0, simplified, OF dist this, of n] dist c-undo-n
have dist-u-ts: simple-ownership-distinct u-ts
by auto
then interpret dist-u-ts-inter: simple-ownership-distinct u-ts **.**

{
fix a
have u-shared a = $\mathcal{S}$ a
proof (cases a ∈ $\mathcal{O}_l$)
case True **with** shared-unch

```
      have u-shared a = 𝒮ₗ a
        by auto
      moreover
      from True shared-orig-unch
      have 𝒮 a = 𝒮ₗ a
        by auto
      ultimately show ?thesis by simp
    next
      case False
      with shared-sim
      show ?thesis
        by auto
    qed
  } hence u-shared-eq: u-shared = 𝒮 by − (rule ext, auto)

  {
   assume safe: map owned u-ts,map released u-ts,i ⊢(is,j,u-m,𝒟,𝒪,u-shared)√
    then have False
    proof cases
      case Read
      then show ?thesis
      using  ts-i tsₗ-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
        by (auto simp add:eqs races-def split: if-split-asm)
    next
      case WriteNonVolatile
      then show ?thesis
      using  ts-i tsₗ-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
        by (auto simp add:eqs races-def split: if-split-asm)
    next
      case WriteVolatile
      then show ?thesis
      using  ts-i tsₗ-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
        apply (auto simp add:eqs races-def split: if-split-asm)
        apply fastforce
        done
    next
      case Fence
      then show ?thesis
      using  ts-i tsₗ-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
        by (auto simp add:eqs races-def split: if-split-asm)
    next
      case Ghost
      then show ?thesis
      using  ts-i tsₗ-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
        apply (auto simp add:eqs races-def split: if-split-asm)
        apply fastforce
        done
    next
      case (RMWReadOnly cond t a′ D f ret A L R W)
      with ts-i is obtain
```

ins: ins = RMW a′ t (D, f) cond ret A L R W **and**
owned-or-shared: a′ ∈ $\mathcal{O}$ ∨ a′ ∈ dom u-shared **and**
cond: ¬ cond (j(t ↦ u-m a′)) **and**
rels-race: ∀ j<length (map owned u-ts). i ≠ j ⟶ ((map released u-ts) ! j) a′ ≠ Some False
by auto
from dist-u-ts-inter.simple-ownership-distinct [OF j-bound-u i-bound-u neq-j-i u-ts-j [simplified ts_l-j]
u-ts-i [simplified ts-i]]
have dist: $\mathcal{O}_l$ ∩ $\mathcal{O}$ = {}
by auto
from owned-or-shared dist a-owned a-unshared shared-orig-unch
have a′-a: a′≠a
by (auto simp add: u-shared-eq domIff)
have u-m-eq: u-m a′ = m a′
proof (cases a′ ∈ $\mathcal{O}_l$)
case True **with** mem-unch
have u-m a′ = m_l a′
by auto
moreover
from True mem-unch-orig a′-a
have m a′ = m_l a′
by auto
ultimately show ?thesis **by** simp
next
case False
with mem-sim
show ?thesis
by auto
qed
with ins cond rels-race **show** ?thesis
using ts-i ts_l-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
by (auto simp add:eqs races-def u-shared-eq u-m-eq split: if-split-asm)
next
case (RMWWrite cond t a′ A L R D f ret W)
with ts-i is **obtain**
ins: ins = RMW a′ t (D, f) cond ret A L R W **and**
cond: cond (j(t ↦ u-m a′)) **and**
a′: ∀ j<length (map owned u-ts). i ≠ j ⟶ a′ ∉ (map owned u-ts) ! j ∪ dom ((map released u-ts) ! j) **and**
safety:
A ⊆ dom u-shared ∪ $\mathcal{O}$ L ⊆ A R ⊆ $\mathcal{O}$ A ∩ R = {}
∀ j<length (map owned u-ts). i ≠ j ⟶ A ∩ ((map owned u-ts) ! j ∪ dom ((map released u-ts) ! j)) = {}
a′ ∉ read-only u-shared
by auto
from a′[rule-format, of j] j-bound-u u-ts-j ts_l-j neq-j-i
have a′ ∉ $\mathcal{O}_l$
by auto
from mem-sim [rule-format, OF this]

```
      have u-m-eq: u-m a′ = m a′
        by auto

      with ins cond safety a′ show ?thesis
      using  ts-i tsₗ-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
        apply (auto simp add:eqs races-def u-shared-eq u-m-eq split: if-split-asm)
        apply fastforce
        done
    next
      case Nil
      then show ?thesis
      using  ts-i tsₗ-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
        by (auto simp add:eqs races-def split: if-split-asm)
    qed
  }
  hence ¬ safe-delayed (u-ts, u-m, u-shared)
    apply (clarsimp simp add: safe-delayed-def)
    apply (rule-tac x=i in exI)
    using u-ts-i ts-i i-bound-u
    apply auto
    done
  moreover
  from safe-delayed-c-undo′ [rule-format, of n] c-undo-n
  have safe-delayed (u-ts, u-m, u-shared)
    by simp
  ultimately have False
    by simp
  thus ?thesis
    by simp
 qed
next
 case WriteVolatile
 with tsₗ′-j tsₗ′ ts-j j-boundₗ have 𝓡ⱼ = Map.empty
   by auto
 with 𝓡ⱼ-non-empty have False by auto
 thus ?thesis ..
next
 case Fence
 with tsₗ′-j tsₗ′ ts-j j-boundₗ have 𝓡ⱼ = Map.empty
   by auto
 with 𝓡ⱼ-non-empty have False by auto
 thus ?thesis ..
next
 case RMWReadOnly
 with tsₗ′-j tsₗ′ ts-j j-boundₗ have 𝓡ⱼ = Map.empty
   by auto
 with 𝓡ⱼ-non-empty have False by auto
 thus ?thesis ..
next
 case RMWWrite
```

with $ts_l{}'$-j $ts_l{}'$ ts-j j-bound$_l$ **have** $\mathcal{R}_j$ = Map.empty
by auto
with $\mathcal{R}_j$-non-empty **have** False **by** auto
thus ?thesis **..**
next
case (Ghost A L R W)
then obtain
is_l: is_l = Ghost A L R W # $is_l{}'$ **and**
$j_l{}'$: $j_l{}' = j_l$ **and**
$sb_l{}'$: $sb_l{}'=sb_l$ **and**
$\mathcal{D}_l{}'$: $\mathcal{D}_l{}'=\mathcal{D}_l$ **and**
$\mathcal{O}_l{}'$: $\mathcal{O}_l{}' = \mathcal{O}_l \cup A - R$ **and**
$\mathcal{R}_l{}'$: $\mathcal{R}_l{}'$= augment-rels (dom $\mathcal{S}_l$) R $\mathcal{R}_l$ **and**
$\mathcal{S}_l{}'$: $\mathcal{S}_l{}'=\mathcal{S}_l \oplus_W R \ominus_A L$ **and**
$m_l{}'$: $m_l{}' = m_l$
by auto
note eqs$'$ = $j_l{}'$ $sb_l{}'$ $\mathcal{D}_l{}'$ $\mathcal{O}_l{}'$ $\mathcal{R}_l{}'$ $\mathcal{S}_l{}'$ $m_l{}'$
from $ts_l{}'$-j $ts_l{}'$ ts-j j-bound$_l$ eqs$'$
obtain eqs: $p_l=p_j$ $is_l{}'=is_j$ $j_l=j_j$ $\mathcal{D}_l=\mathcal{D}_j$ $\mathcal{O}_l \cup A - R = \mathcal{O}_j$
augment-rels (dom $\mathcal{S}_l$) R $\mathcal{R}_l=\mathcal{R}_j$
by auto

from safe$_l$ [simplified is_l]
obtain
A-shared-owned: A $\subseteq$ dom $\mathcal{S}_l \cup \mathcal{O}_l$ **and** L-A: L $\subseteq$ A **and** R-owns: R $\subseteq \mathcal{O}_l$ **and** A-R: A $\cap$ R = {} **and**
$\forall j' <$ length (map owned ts_l). $j \neq j' \longrightarrow$ A $\cap$ ((map owned ts_l)!j' $\cup$ dom ((map released ts_l)!j')) = {}
by cases auto

from A-shared-owned L-A R-owns A-R
have shared-eq: $\forall$a. a $\notin \mathcal{O}_l \longrightarrow$ a $\notin \mathcal{O}_l{}' \longrightarrow \mathcal{S}_l$ a = ($\mathcal{S}_l \oplus_W R \ominus_A L$) a
by (auto simp add: restrict-shared-def augment-shared-def $\mathcal{O}_l{}'$ split: option.splits)

from undo-local-steps [**where** c=c, OF trace-rest c-last$'$ idle-rest$'$ safe-delayed-last, simplified $ts_l{}'$,
simplified,
OF j-bound$_l$ ts_l-j [simplified], simplified $m_l{}'$ $\mathcal{S}_l{}'$, simplified,
OF shared-eq dist-last dist-last$'$ [simplified $ts_l{}'$,simplified] safe-delayed-upto-last]

obtain c$'$ k **where**
k-bound: k $\leq$ n $-$ last **and**
trace-c$'$: trace c$'$ (Suc last) k **and**
c$'$-first: c$'$ (Suc last) = (ts_l, m_l, $\mathcal{S}_l$) **and**
c$'$-leq: ($\forall$x$\leq$k. length (fst (c$'$ (Suc (last + x)))) = length (fst (c (Suc (last + x))))) **and**
c$'$-safe: ($\forall$x<k. safe-delayed (c$'$ (Suc (last + x)))) **and**
c$'$-unsafe: (k < n $-$ last $\longrightarrow \neg$ safe-delayed (c$'$ (Suc (last + k)))) **and**
c$'$-unch:

$(\forall x \le k.\ \forall ts_x\ \mathcal{S}_x\ m_x.$
$c\ (Suc\ (last + x)) = (ts_x, m_x, \mathcal{S}_x) \longrightarrow$
$(\forall ts_x{}'\ \mathcal{S}_x{}'\ m_x{}'.$
$c'\ (Suc\ (last + x)) = (ts_x{}', m_x{}', \mathcal{S}_x{}') \longrightarrow$
$ts_x{}'\ !\ j = ts_l\ !\ j\ \wedge$
$(\forall a \in \mathcal{O}_l.\ \mathcal{S}_x{}'\ a = \mathcal{S}_l\ a)\ \wedge$
$(\forall a \in \mathcal{O}_l.\ \mathcal{S}_x\ a = (\mathcal{S}_l \oplus_W R \ominus_A L)\ a)\ \wedge$
$(\forall a \in \mathcal{O}_l.\ m_x{}'\ a = m_l\ a) \wedge (\forall a' \in \mathcal{O}_l.\ m_x\ a' = (m_l)\ a')))$ **and**

c′-sim:
$(\forall x \le k.\ \forall ts_x\ \mathcal{S}_x\ m_x.$
$c\ (Suc\ (last + x)) = (ts_x, m_x, \mathcal{S}_x) \longrightarrow$
$(\forall ts_x{}'\ \mathcal{S}_x{}'\ m_x{}'.$
$c'\ (Suc\ (last + x)) = (ts_x{}', m_x{}', \mathcal{S}_x{}') \longrightarrow$
$(\forall ja < length\ ts_x.\ ja \neq j \longrightarrow ts_x{}'\ !\ ja = ts_x\ !\ ja)\ \wedge$
$(\forall a.\ a \notin \mathcal{O}_l \longrightarrow a \notin \mathcal{O}_l{}' \longrightarrow \mathcal{S}_x{}'\ a = \mathcal{S}_x\ a)\ \wedge$
$(\forall a.\ a \notin \mathcal{O}_l \longrightarrow m_x{}'\ a = m_x\ a)))$

by (clarsimp)
obtain c-undo **where** c-undo: c-undo = (λx. if x $\le$ last then c x else c′ (Suc last + x − last))
by blast
have c-undo-0: c-undo 0 = c_0
by (auto simp add: c-undo c-0)
from sequence-traces [OF trace-last trace-c′, simplified c-last, OF c′-first c-undo]
have trace-undo: trace c-undo 0 (last + k) **.**
obtain u-ts u-shared u-m **where**
c-undo-n: c-undo n = (u-ts,u-m, u-shared)
by (cases c-undo n)
with last-bound c′-first c-last
have c′-suc: c′ (Suc n) = (u-ts,u-m, u-shared)
apply (auto simp add: c-undo split: if-split-asm)
apply (subgoal-tac n=last)
apply auto
done

show ?thesis
proof (cases k < n − last)
case True
with c′-unsafe **have** unsafe: ¬ safe-delayed (c-undo (last + k))
by (auto simp add: c-undo c-last c′-first)
from True **have** last + k $\le$ n
by auto
from safe-delayed-reach-inter.safe-config [OF this trace-undo, of last + k]
have safe-delayed (c-undo (last + k))
by (auto simp add: c-undo c-0)
with unsafe **have** False **by** simp
thus ?thesis **..**
next
case False

with k-bound **have** k: k = n − last
  by auto
have eq′: Suc (last + (n − last)) = Suc n
  using last-bound
  by simp
from c′-unch [rule-format, of k, simplified k eq′, OF - c-suc c′-suc]
obtain u-ts-j: u-ts!j = ts_l!j **and**
  shared-unch: ∀a∈$\mathcal{O}_l$. u-shared a = $\mathcal{S}_l$ a **and**
  shared-orig-unch: ∀a∈$\mathcal{O}_l$. $\mathcal{S}$ a = ($\mathcal{S}_l$ $\oplus_W$ R $\ominus_A$ L) a **and**
  mem-unch: ∀a∈$\mathcal{O}_l$. u-m a = m_l a **and**
  mem-unch-orig: ∀a′∈$\mathcal{O}_l$. m a′ = m_l a′
  by auto

from c′-sim [rule-format, of k, simplified k eq′, OF - c-suc c′-suc] i-bound neq-j-i
obtain u-ts-i: u-ts!i = ts!i **and**
  shared-sim: ∀a. a ∉ $\mathcal{O}_l$ ⟶ a ∉ $\mathcal{O}_l$′ ⟶ u-shared a = $\mathcal{S}$ a **and**
  mem-sim: ∀a. a ∉ $\mathcal{O}_l$ ⟶ u-m a = m a
  by auto

from c′-leq [rule-format, of k] c′-suc c-suc
have leq-u-ts: length u-ts = length ts
  by (auto simp add: eq′ k)

from j-bound leq-u-ts
have j-bound-u: j < length u-ts
  by simp
from i-bound leq-u-ts
have i-bound-u: i < length u-ts
  by simp
from k last-bound **have** l-k-eq: last + k = n
  by auto
from safe-delayed-reach-inter.safe-config [OF - trace-undo, simplified l-k-eq]
  k c-0 last-bound
have safe-delayed-c-undo′: ∀x≤n. safe-delayed (c-undo x)
  by (auto simp add: c-undo split: if-split-asm)
hence safe-delayed-c-undo: ∀x<n. safe-delayed (c-undo x)
  by auto
from trace-preserves-simple-ownership-distinct [OF - trace-undo,
  simplified l-k-eq c-undo-0, simplified, OF dist this, of n] dist c-undo-n
have dist-u-ts: simple-ownership-distinct u-ts
  by auto
then interpret dist-u-ts-inter: simple-ownership-distinct u-ts **.**
{
  fix a
  have u-m a = m a
  proof (cases a ∈ $\mathcal{O}_l$)
    case True **with** mem-unch
    have u-m a = m_l a
      by auto
    moreover

from True mem-unch-orig
have m a = m_l a
by auto
ultimately show ?thesis **by** simp
next
case False
with mem-sim
show ?thesis
by auto
qed
} **hence** u-m-eq: u-m = m **by** − (rule ext, auto)
{
assume safe: map owned u-ts,map released u-ts,i ⊢(is,j,u-m,$\mathcal{D}$,$\mathcal{O}$,u-shared)$\surd$
then have False
proof cases
case (Read a volatile t)
with ts-i is **obtain**
ins: ins = Read volatile a t **and**
access-cond: a ∈ $\mathcal{O}$ ∨ a ∈ read-only u-shared ∨ volatile ∧ a ∈ dom u-shared **and**
rels-cond: ∀ j<length u-ts. i ≠ j ⟶ ((map released u-ts) ! j) a ≠ Some False **and**
rels-non-volatile-cond: ¬ volatile ⟶ (∀ j<length u-ts. i ≠ j ⟶ a ∉ dom ((map released u-ts) ! j)) **and**
clean: volatile ⟶ ¬ $\mathcal{D}$
by auto

from race ts-j
have rc: augment-rels (dom $\mathcal{S}_l$) R $\mathcal{R}_l$ a = Some False ∨
(¬ volatile ∧ a ∈ dom (augment-rels (dom $\mathcal{S}_l$) R $\mathcal{R}_l$))
by (auto simp add: races-def ins eqs)
from rels-cond [rule-format, simplified, OF j-bound-u neq-j-i [symmetric]] u-ts-j ts_l-j j-bound-u
have $\mathcal{R}_l$-a: $\mathcal{R}_l$ a ≠ Some False
by auto
from dist-u-ts-inter.simple-ownership-distinct [OF j-bound-u i-bound-u neq-j-i u-ts-j [simplified ts_l-j]
u-ts-i [simplified ts-i]]
have dist: $\mathcal{O}_l$ ∩ $\mathcal{O}$ = {}
by auto

show ?thesis
proof (cases volatile)
case True
note volatile=this
show ?thesis
proof (cases a ∈ R)
case False
with rc $\mathcal{R}_l$-a **show** False
by (auto simp add: augment-rels-def volatile)

next
case True
with R-owns
have a-owns$_{\mathsf{l}}$: a ∈ $\mathcal{O}_{\mathsf{l}}$
by auto
from shared-unch [rule-format, OF a-owns$_{\mathsf{l}}$]
have u-shared-eq: u-shared a = $\mathcal{S}_{\mathsf{l}}$ a
by auto
from a-owns$_{\mathsf{l}}$ dist **have** a ∉ $\mathcal{O}$
by auto
moreover
{
assume a ∈ read-only u-shared
with u-shared-eq **have** $\mathcal{S}_{\mathsf{l}}$ a = Some False
by (auto simp add: read-only-def)
with rc True $\mathcal{R}_{\mathsf{l}}$-a **have** False
by (auto simp add: augment-rels-def split: option.splits simp add: domIff volatile)
}
moreover
{
assume a ∈ dom u-shared
with u-shared-eq rc True $\mathcal{R}_{\mathsf{l}}$-a **have** False
by (auto simp add: augment-rels-def split: option.splits simp add: domIff volatile)
}
ultimately show False
using access-cond
by auto
qed
next
case False
note non-volatile = this
from rels-non-volatile-cond [rule-format, OF False j-bound-u neq-j-i [symmetric]] u-ts-j ts$_{\mathsf{l}}$-j j-bound-u
have $\mathcal{R}_{\mathsf{l}}$-a: $\mathcal{R}_{\mathsf{l}}$ a = None
by (auto simp add: domIff)
show ?thesis
proof (cases a ∈ R)
case False
with rc $\mathcal{R}_{\mathsf{l}}$-a **show** False
by (auto simp add: augment-rels-def non-volatile domIff)
next
case True
with R-owns
have a-owns$_{\mathsf{l}}$: a ∈ $\mathcal{O}_{\mathsf{l}}$
by auto
from shared-unch [rule-format, OF a-owns$_{\mathsf{l}}$]
have u-shared-eq: u-shared a = $\mathcal{S}_{\mathsf{l}}$ a
by auto

from a-owns$_\mathsf{l}$ dist **have** a-unowned: a $\notin$ $\mathcal{O}$
by auto
moreover
from ro-last-last **interpret**
read-only-unowned $\mathcal{S}_\mathsf{l}$ ts$_\mathsf{l}$ **.**
from read-only-unowned [OF j-bound$_\mathsf{l}$ ts$_\mathsf{l}$-j] a-owns$_\mathsf{l}$ **have** a-unsh: a $\notin$ read-only $\mathcal{S}_\mathsf{l}$ **by** auto
{
assume a $\in$ read-only u-shared
with u-shared-eq **have** sh: $\mathcal{S}_\mathsf{l}$ a = Some False
by (auto simp add: read-only-def)

with rc True $\mathcal{R}_\mathsf{l}$-a access-cond u-shared-eq a-unowned sh a-owns$_\mathsf{l}$ a-unsh **have** False
by (auto simp add: augment-rels-def split: option.splits simp add: domIff non-volatile read-only-def)
}
moreover
{
assume a $\in$ dom u-shared
with u-shared-eq rc True $\mathcal{R}_\mathsf{l}$-a a-owns$_\mathsf{l}$ a-unsh access-cond dist **have** False
by (auto simp add: augment-rels-def split: option.splits simp add: domIff non-volatile read-only-def)
}
ultimately show False
using access-cond
by (auto)
qed
qed
next
case (WriteNonVolatile a D f A$'$ L$'$ R$'$ W$'$)
with ts-i is **obtain**
ins: ins = Write False a (D, f) A$'$ L$'$ R$'$ W$'$ **and**
a-owned: a $\in$ $\mathcal{O}$ **and** a-unshared: a $\notin$ dom u-shared **and**
a-unreleased: $\forall$j<length u-ts. i $\neq$ j $\longrightarrow$ a $\notin$ dom ((map released u-ts) ! j)
by auto
from dist-u-ts-inter.simple-ownership-distinct [OF j-bound-u i-bound-u neq-j-i u-ts-j [simplified ts$_\mathsf{l}$-j]
u-ts-i [simplified ts-i]]
have dist: $\mathcal{O}_\mathsf{l}$ $\cap$ $\mathcal{O}$ = {}
by auto
from race ts-j
have rc: a $\in$ dom (augment-rels (dom $\mathcal{S}_\mathsf{l}$) R $\mathcal{R}_\mathsf{l}$)
by (auto simp add: races-def ins eqs)
from a-unreleased [rule-format, simplified, OF j-bound-u neq-j-i [symmetric]] u-ts-j ts$_\mathsf{l}$-j j-bound-u
have $\mathcal{R}_\mathsf{l}$-a: a $\notin$ dom $\mathcal{R}_\mathsf{l}$
by auto
show False

proof (cases $a \in R$)
case False
with rc $\mathcal{R}_{\mathsf{l}}$-a **show** False
by (auto simp add: augment-rels-def domIff)
next
case True
with R-owns
have a-owns$_{\mathsf{l}}$: $a \in \mathcal{O}_{\mathsf{l}}$
by auto
with a-owned dist **show** False
by auto
qed
next
case (WriteVolatile a A' L' R' D f W')
with ts-i is **obtain**
ins: ins = Write True a (D, f) A' L' R' W' **and**
a-un-owned-released: $\forall$j<length u-ts. i $\neq$ j $\longrightarrow$
a $\notin$ ((map owned u-ts) ! j) $\wedge$ a $\notin$ dom ((map released u-ts) ! j) **and**
A'-owns-shared: $A' \subseteq$ dom u-shared $\cup$ $\mathcal{O}$ **and**
L'-A': $L' \subseteq A'$ **and**
R'-owned: $R' \subseteq \mathcal{O}$ **and**
A'-R': $A' \cap R' = \{\}$ **and**
acq-ok: $\forall$j<length u-ts. i $\neq$ j $\longrightarrow$ $A' \cap$ ((map owned u-ts) ! j $\cup$ dom ((map released u-ts) ! j)) = {} **and**

writeable: a $\notin$ read-only u-shared
by auto
from a-un-owned-released [rule-format, simplified, OF j-bound-u neq-j-i [symmetric]] u-ts-j ts$_{\mathsf{l}}$-j j-bound-u
obtain $\mathcal{O}_{\mathsf{l}}$-a: $a \notin \mathcal{O}_{\mathsf{l}}$ **and** $\mathcal{R}_{\mathsf{l}}$-a: a $\notin$ dom ($\mathcal{R}_{\mathsf{l}}$)
by auto
from acq-ok [rule-format, simplified, OF j-bound-u neq-j-i [symmetric]] u-ts-j ts$_{\mathsf{l}}$-j j-bound-u
obtain $\mathcal{O}_{\mathsf{l}}$-A': $A' \cap \mathcal{O}_{\mathsf{l}} = \{\}$ **and** $\mathcal{R}_{\mathsf{l}}$-A': $A' \cap$ dom ($\mathcal{R}_{\mathsf{l}}$) = {}
by auto
{
assume rc: $a \in$ dom (augment-rels (dom $\mathcal{S}_{\mathsf{l}}$) R $\mathcal{R}_{\mathsf{l}}$)
have False
proof (cases $a \in R$)
case False
with rc $\mathcal{R}_{\mathsf{l}}$-a **show** False
by (auto simp add: augment-rels-def domIff)
next
case True
with R-owns
have a-owns$_{\mathsf{l}}$: $a \in \mathcal{O}_{\mathsf{l}}$
by auto
with $\mathcal{O}_{\mathsf{l}}$-a **show** False
by auto
qed

}
moreover
{
assume rc: $A' \cap$ dom (augment-rels (dom $\mathcal{S}_{\mathsf{l}}$) R $\mathcal{R}_{\mathsf{l}}$) $\neq$ {}
then obtain a' **where** a'-A': $a' \in A'$ **and** a'-aug: $a' \in$ dom (augment-rels (dom $\mathcal{S}_{\mathsf{l}}$) R $\mathcal{R}_{\mathsf{l}}$)
by auto
have False
proof (cases $a' \in$ R)
case False
with a'-aug a'-A' $\mathcal{R}_{\mathsf{l}}$-A' **show** False
by (auto simp add: augment-rels-def domIff)
next
case True
with R-owns **have** a'-owns$_{\mathsf{l}}$: $a' \in \mathcal{O}_{\mathsf{l}}$
by auto
with $\mathcal{O}_{\mathsf{l}}$-A' a'-A' **show** False
by auto
qed
}
ultimately show False
using race ts-j
by (auto simp add: races-def ins eqs)
next
case Fence
then show ?thesis
using ts-i ts$_{\mathsf{l}}$-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
by (auto simp add:eqs races-def split: if-split-asm)
next
case (Ghost A' L' R' W')
with ts-i is **obtain**
ins: ins = Ghost A' L' R' W' **and**
A'-owns-shared: $A' \subseteq$ dom u-shared $\cup$ $\mathcal{O}$ **and**
L'-A': $L' \subseteq A'$ **and**
R'-owned: $R' \subseteq \mathcal{O}$ **and**
A'-R': $A' \cap R' =$ {} **and**
acq-ok: $\forall$j<length u-ts. i $\neq$ j $\longrightarrow$ $A' \cap$ ((map owned u-ts) ! j $\cup$ dom ((map released u-ts) ! j)) = {}
by auto
from acq-ok [rule-format, simplified, OF j-bound-u neq-j-i [symmetric]] u-ts-j ts$_{\mathsf{l}}$-j j-bound-u
obtain $\mathcal{O}_{\mathsf{l}}$-A': $A' \cap \mathcal{O}_{\mathsf{l}}$ = {} **and** $\mathcal{R}_{\mathsf{l}}$-A': $A' \cap$ dom ($\mathcal{R}_{\mathsf{l}}$) = {}
by auto

from race ts-j
obtain a' **where** a'-A': $a' \in A'$ **and**
a'-aug: $a' \in$ dom (augment-rels (dom $\mathcal{S}_{\mathsf{l}}$) R $\mathcal{R}_{\mathsf{l}}$)
by (auto simp add: races-def ins eqs)
show False
proof (cases $a' \in$ R)

case False
with a′-aug a′-A′ $\mathcal{R}_{\mathsf{l}}$-A′ **show** False
by (auto simp add: augment-rels-def domIff)
next
case True
with R-owns **have** a′-owns$_{\mathsf{l}}$: a′ ∈ $\mathcal{O}_{\mathsf{l}}$
by auto
with $\mathcal{O}_{\mathsf{l}}$-A′ a′-A′ **show** False
by auto
qed
next
case (RMWReadOnly cond t a D f ret A′ L′ R′ W′)
with ts-i is **obtain**
ins: ins = RMW a t (D, f) cond ret A′ L′ R′ W′ **and**
owned-or-shared: a ∈ $\mathcal{O}$ ∨ a ∈ dom u-shared **and**
cond: ¬ cond (j(t ↦ u-m a)) **and**
rels-race: ∀ j<length (map owned u-ts). i ≠ j ⟶ ((map released u-ts) ! j) a ≠ Some False
by auto
from dist-u-ts-inter.simple-ownership-distinct [OF j-bound-u i-bound-u neq-j-i u-ts-j [simplified ts$_{\mathsf{l}}$-j]
u-ts-i [simplified ts-i]]
have dist: $\mathcal{O}_{\mathsf{l}}$ ∩ $\mathcal{O}$ = {}
by auto
from race ts-j cond
have rc: augment-rels (dom $\mathcal{S}_{\mathsf{l}}$) R $\mathcal{R}_{\mathsf{l}}$ a = Some False
by (auto simp add: races-def ins eqs u-m-eq)

from rels-race [rule-format, simplified, OF j-bound-u neq-j-i [symmetric]]
u-ts-j ts$_{\mathsf{l}}$-j j-bound-u
have $\mathcal{R}_{\mathsf{l}}$-a: $\mathcal{R}_{\mathsf{l}}$ a ≠ Some False
by auto

show ?thesis
proof (cases a ∈ R)
case False
with rc $\mathcal{R}_{\mathsf{l}}$-a **show** False
by (auto simp add: augment-rels-def)
next
case True
with R-owns
have a-owns$_{\mathsf{l}}$: a ∈ $\mathcal{O}_{\mathsf{l}}$
by auto
from shared-unch [rule-format, OF a-owns$_{\mathsf{l}}$]
have u-shared-eq: u-shared a = $\mathcal{S}_{\mathsf{l}}$ a
by auto
from a-owns$_{\mathsf{l}}$ dist **have** a ∉ $\mathcal{O}$
by auto
with u-shared-eq rc True $\mathcal{R}_{\mathsf{l}}$-a owned-or-shared **show** False
by (auto simp add: augment-rels-def split: option.splits simp add: domIff)

qed
next
case (RMWWrite cond t a A′ L′ R′ D f ret W′)
with ts-i is **obtain**
ins: ins = RMW a t (D, f) cond ret A′ L′ R′ W′ **and**
cond: cond (j(t ↦ u-m a)) **and**
a-un-owned-released: ∀j<length (map owned u-ts). i ≠ j ⟶ a ∉ (map owned u-ts) ! j ∪ dom ((map released u-ts) ! j) **and**
A′-owns-shared:A′ ⊆ dom u-shared ∪ $\mathcal{O}$ **and**
L′-A′: L′ ⊆ A′ **and**
R′-owned: R′ ⊆ $\mathcal{O}$ **and**
A′-R′: A′ ∩ R′ = {} **and**
acq-ok: ∀j<length (map owned u-ts). i ≠ j ⟶ A′ ∩ ((map owned u-ts) ! j ∪ dom ((map released u-ts) ! j)) = {} **and**
writeable: a ∉ read-only u-shared
by auto

from a-un-owned-released [rule-format, simplified, OF j-bound-u neq-j-i [symmetric]] u-ts-j ts$_{\mathsf{l}}$-j j-bound-u
obtain $\mathcal{O}_{\mathsf{l}}$-a: a ∉ $\mathcal{O}_{\mathsf{l}}$ **and** $\mathcal{R}_{\mathsf{l}}$-a: a ∉ dom ($\mathcal{R}_{\mathsf{l}}$)
by auto
from acq-ok [rule-format, simplified, OF j-bound-u neq-j-i [symmetric]] u-ts-j ts$_{\mathsf{l}}$-j j-bound-u
obtain $\mathcal{O}_{\mathsf{l}}$-A′: A′ ∩ $\mathcal{O}_{\mathsf{l}}$ = {} **and** $\mathcal{R}_{\mathsf{l}}$-A′: A′ ∩ dom ($\mathcal{R}_{\mathsf{l}}$) = {}
by auto
{
assume rc: a ∈ dom (augment-rels (dom $\mathcal{S}_{\mathsf{l}}$) R $\mathcal{R}_{\mathsf{l}}$)
have False
proof (cases a ∈ R)
case False
with rc $\mathcal{R}_{\mathsf{l}}$-a **show** False
by (auto simp add: augment-rels-def domIff)
next
case True
with R-owns
have a-owns$_{\mathsf{l}}$: a ∈ $\mathcal{O}_{\mathsf{l}}$
by auto
with $\mathcal{O}_{\mathsf{l}}$-a **show** False
by auto
qed
}
moreover
{
assume rc: A′ ∩ dom (augment-rels (dom $\mathcal{S}_{\mathsf{l}}$) R $\mathcal{R}_{\mathsf{l}}$) ≠ {}
then obtain a′ **where** a′-A′: a′ ∈ A′ **and** a′-aug: a′ ∈ dom (augment-rels (dom $\mathcal{S}_{\mathsf{l}}$) R $\mathcal{R}_{\mathsf{l}}$)
by auto
have False
proof (cases a′ ∈ R)

```
            case False
            with a'-aug a'-A' 𝓡𝗅-A' show False
              by (auto simp add: augment-rels-def domIff)
          next
            case True
            with R-owns have a'-owns𝗅: a' ∈ 𝓞𝗅
              by auto
            with 𝓞𝗅-A' a'-A' show False
              by auto
          qed
        }
        ultimately show False
        using race ts-j cond
          by (auto simp add: races-def ins eqs u-m-eq)
      next
      next
        case Nil
        then show ?thesis
        using  ts-i ts𝗅-j race is j-bound i-bound u-ts-i u-ts-j leq-u-ts neq-j-i ts-j
          by (auto simp add:eqs races-def split: if-split-asm)
      qed
    }
    hence ¬ safe-delayed (u-ts, u-m, u-shared)
      apply (clarsimp simp add: safe-delayed-def)
      apply (rule-tac x=i in exI)
      using u-ts-i ts-i i-bound-u
      apply auto
      done
    moreover
    from safe-delayed-c-undo' [rule-format, of n] c-undo-n
    have safe-delayed (u-ts, u-m, u-shared)
      by simp
    ultimately have False
      by simp
    thus ?thesis
      by simp
  qed
 qed
next
 case (StoreBuffer - p is j sb 𝓓 𝓞 𝓡 sb' 𝓞' 𝓡')
 hence False
   by (auto simp add: empty-storebuffer-step-def)
 thus ?thesis ..
qed
qed
}
ultimately show ?thesis
using last-action-of-thread [where i=j, OF trace]
  by blast
qed
```

qed

datatype $'$p memref =
  Write$_{sb}$ bool addr sop val acq lcl rel wrt
| Read$_{sb}$ bool addr tmp val
| Prog$_{sb}$ $'$p $'$p instrs
| Ghost$_{sb}$ acq lcl rel wrt

type-synonym $'$p store-buffer = $'$p memref list
inductive flush-step:: memory × $'$p store-buffer × owns × rels × shared ⇒ memory × $'$p store-buffer × owns × rels × shared ⇒ bool
(‹- $\to_f$ -› [60,60] 100)
where
Write$_{sb}$: ⟦$\mathcal{O}'$ = (if volatile then $\mathcal{O} \cup$ A − R else $\mathcal{O}$);
$\mathcal{S}'$ = (if volatile then $\mathcal{S} \oplus_W$ R $\ominus_A$ L else $\mathcal{S}$);
$\mathcal{R}'$=(if volatile then Map.empty else $\mathcal{R}$)⟧
⟹
(m, Write$_{sb}$ volatile a sop v A L R W# rs,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\to_f$ (m(a := v), rs,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$)
| Read$_{sb}$: (m, Read$_{sb}$ volatile a t v#rs,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\to_f$ (m, rs,$\mathcal{O}$,$\mathcal{R}$, $\mathcal{S}$)
| Prog$_{sb}$: (m, Prog$_{sb}$ p p$'$ is#rs,$\mathcal{O}$,$\mathcal{R}$, $\mathcal{S}$) $\to_f$ (m, rs,$\mathcal{O}$,$\mathcal{R}$, $\mathcal{S}$)
| Ghost: (m, Ghost$_{sb}$ A L R W# rs,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\to_f$ (m, rs,$\mathcal{O} \cup$ A − R, augment-rels (dom $\mathcal{S}$) R $\mathcal{R}$, $\mathcal{S} \oplus_W$ R $\ominus_A$ L)

abbreviation flush-steps::memory × $'$p store-buffer × owns × rels × shared ⇒ memory × $'$p store-buffer × owns × rels × shared⇒ bool
(‹- $\to_f{}^*$ -› [60,60] 100)
where
flush-steps == flush-step^**

term x $\to_f{}^*$ Y

lemmas flush-step-induct =
flush-step.induct [split-format (complete),
consumes 1, case-names Write$_{sb}$ Read$_{sb}$ Prog$_{sb}$ Ghost]

inductive store-buffer-step:: memory × $'$p store-buffer × $'$owns × $'$rels × $'$shared ⇒ memory × $'$p memref list × $'$owns × $'$rels × $'$shared ⇒ bool
(‹- $\to_w$ -› [60,60] 100)
where
SBWrite$_{sb}$:
(m, Write$_{sb}$ volatile a sop v A L R W# rs,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\to_w$ (m(a := v), rs,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$)

abbreviation store-buffer-steps::memory × $'$p store-buffer × $'$owns × $'$rels × $'$shared ⇒ memory × $'$p store-buffer × $'$owns × $'$rels × $'$shared⇒ bool
(‹- →w* -› [60,60] 100)
where
store-buffer-steps == store-buffer-step^**

term x →w* Y

fun buffered-val :: 'p memref list ⇒ addr ⇒ val option
where
  buffered-val [] a = None
| buffered-val (r # rs) a′ =
   (case r of
      Write_{sb} volatile a - v - - - - ⇒ (case buffered-val rs a′ of
                              None ⇒ (if a′=a then Some v else None)
                            | Some v′ ⇒ Some v′)
     | - ⇒ buffered-val rs a′)

definition address-of :: 'p memref ⇒ addr set
where
address-of r = (case r of Write_{sb} volatile a - v - - - - ⇒ {a} | Read_{sb} volatile a t v ⇒ {a} |
              - ⇒ {})

lemma address-of-simps [simp]:
address-of (Write_{sb} volatile a sop v A L R W) = {a}
address-of (Read_{sb} volatile a t v) = {a}
address-of (Prog_{sb} p p′ is) = {}
address-of (Ghost_{sb} A L R W) = {}
  by (auto simp add: address-of-def)

definition is-volatile :: 'p memref ⇒ bool
where
is-volatile r = (case r of Write_{sb} volatile a - v - - - -⇒ volatile | Read_{sb} volatile a t v ⇒ volatile
 | - ⇒ False)

lemma is-volatile-simps [simp]:
is-volatile (Write_{sb} volatile a sop v A L R W) = volatile
is-volatile (Read_{sb} volatile a t v) = volatile
is-volatile (Prog_{sb} p p′ is) = False
is-volatile (Ghost_{sb} A L R W) = False
  by (auto simp add: is-volatile-def)

definition is-Write_{sb}:: 'p memref ⇒ bool
where
is-Write_{sb} r = (case r of Write_{sb} volatile a - v - - - -⇒ True | - ⇒ False)

definition is-Read_{sb}:: 'p memref ⇒ bool
where
is-Read_{sb} r = (case r of Read_{sb} volatile a t v ⇒ True | - ⇒ False)

definition is-Prog_{sb}:: 'p memref ⇒ bool
where
is-Prog_{sb} r = (case r of Prog_{sb} - - - ⇒ True | - ⇒ False)

definition is-Ghost_{sb}:: 'p memref ⇒ bool
where
is-Ghost_{sb} r = (case r of Ghost_{sb} - - - - ⇒ True | - ⇒ False)

lemma is-Write$_{\mathsf{sb}}$-simps [simp]:
is-Write$_{\mathsf{sb}}$ (Write$_{\mathsf{sb}}$ volatile a sop v A L R W) = True
is-Write$_{\mathsf{sb}}$ (Read$_{\mathsf{sb}}$ volatile a t v) = False
is-Write$_{\mathsf{sb}}$ (Prog$_{\mathsf{sb}}$ p p' is) = False
is-Write$_{\mathsf{sb}}$ (Ghost$_{\mathsf{sb}}$ A L R W) = False
by (auto simp add: is-Write$_{\mathsf{sb}}$-def)

lemma is-Read$_{\mathsf{sb}}$-simps [simp]:
is-Read$_{\mathsf{sb}}$ (Read$_{\mathsf{sb}}$ volatile a t v) = True
is-Read$_{\mathsf{sb}}$ (Write$_{\mathsf{sb}}$ volatile a sop v A L R W) = False
is-Read$_{\mathsf{sb}}$ (Prog$_{\mathsf{sb}}$ p p' is) = False
is-Read$_{\mathsf{sb}}$ (Ghost$_{\mathsf{sb}}$ A L R W) = False
by (auto simp add: is-Read$_{\mathsf{sb}}$-def)

lemma is-Prog$_{\mathsf{sb}}$-simps [simp]:
is-Prog$_{\mathsf{sb}}$ (Read$_{\mathsf{sb}}$ volatile a t v) = False
is-Prog$_{\mathsf{sb}}$ (Write$_{\mathsf{sb}}$ volatile a sop v A L R W) = False
is-Prog$_{\mathsf{sb}}$ (Prog$_{\mathsf{sb}}$ p p' is) = True
is-Prog$_{\mathsf{sb}}$ (Ghost$_{\mathsf{sb}}$ A L R W) = False
by (auto simp add: is-Prog$_{\mathsf{sb}}$-def)

lemma is-Ghost$_{\mathsf{sb}}$-simps [simp]:
is-Ghost$_{\mathsf{sb}}$ (Read$_{\mathsf{sb}}$ volatile a t v) = False
is-Ghost$_{\mathsf{sb}}$ (Write$_{\mathsf{sb}}$ volatile a sop v A L R W) = False
is-Ghost$_{\mathsf{sb}}$ (Prog$_{\mathsf{sb}}$ p p' is) = False
is-Ghost$_{\mathsf{sb}}$ (Ghost$_{\mathsf{sb}}$ A L R W) = True
by (auto simp add: is-Ghost$_{\mathsf{sb}}$-def)

definition is-volatile-Write$_{\mathsf{sb}}$:: 'p memref ⇒ bool
where
is-volatile-Write$_{\mathsf{sb}}$ r = (case r of Write$_{\mathsf{sb}}$ volatile a - v - - - -⇒ volatile | - ⇒ False)

lemma is-volatile-Write$_{\mathsf{sb}}$-simps [simp]:
is-volatile-Write$_{\mathsf{sb}}$ (Write$_{\mathsf{sb}}$ volatile a sop v A L R W) = volatile
is-volatile-Write$_{\mathsf{sb}}$ (Read$_{\mathsf{sb}}$ volatile a t v) = False
is-volatile-Write$_{\mathsf{sb}}$ (Prog$_{\mathsf{sb}}$ p p' is) = False
is-volatile-Write$_{\mathsf{sb}}$ (Ghost$_{\mathsf{sb}}$ A L R W) = False
by (auto simp add: is-volatile-Write$_{\mathsf{sb}}$-def)

lemma is-volatile-Write$_{\mathsf{sb}}$-address-of [simp]: is-volatile-Write$_{\mathsf{sb}}$ x ⟹ address-of x ≠ {}
by (cases x) auto

definition is-volatile-Read$_{\mathsf{sb}}$:: 'p memref ⇒ bool
where
is-volatile-Read$_{\mathsf{sb}}$ r = (case r of Read$_{\mathsf{sb}}$ volatile a t v ⇒ volatile | - ⇒ False)

lemma is-volatile-Read$_{\mathsf{sb}}$-simps [simp]:
is-volatile-Read$_{\mathsf{sb}}$ (Read$_{\mathsf{sb}}$ volatile a t v) = volatile
is-volatile-Read$_{\mathsf{sb}}$ (Write$_{\mathsf{sb}}$ volatile a sop v A L R W) = False

is-volatile-Read$_{sb}$ (Prog$_{sb}$ p p′ is) = False
is-volatile-Read$_{sb}$ (Ghost$_{sb}$ A L R W) = False
  by (auto simp add: is-volatile-Read$_{sb}$-def)

definition is-non-volatile-Write$_{sb}$:: ′p memref ⇒ bool
where
is-non-volatile-Write$_{sb}$ r = (case r of Write$_{sb}$ volatile a - v - - - -⇒ ¬ volatile | - ⇒ False)

lemma is-non-volatile-Write$_{sb}$-simps [simp]:
is-non-volatile-Write$_{sb}$ (Write$_{sb}$ volatile a sop v A L R W) = (¬ volatile)
is-non-volatile-Write$_{sb}$ (Read$_{sb}$ volatile a t v) = False
is-non-volatile-Write$_{sb}$ (Prog$_{sb}$ p p′ is) = False
is-non-volatile-Write$_{sb}$ (Ghost$_{sb}$ A L R W) = False
  by (auto simp add: is-non-volatile-Write$_{sb}$-def)

definition is-non-volatile-Read$_{sb}$:: ′p memref ⇒ bool
where
is-non-volatile-Read$_{sb}$ r = (case r of Read$_{sb}$ volatile a t v ⇒ ¬ volatile | - ⇒ False)

lemma is-non-volatile-Read$_{sb}$-simps [simp]:
is-non-volatile-Read$_{sb}$ (Read$_{sb}$ volatile a t v) = (¬ volatile)
is-non-volatile-Read$_{sb}$ (Write$_{sb}$ volatile a sop v A L R W) = False
is-non-volatile-Read$_{sb}$ (Prog$_{sb}$ p p′ is) = False
is-non-volatile-Read$_{sb}$ (Ghost$_{sb}$ A L R W) = False
  by (auto simp add: is-non-volatile-Read$_{sb}$-def)

lemma is-volatile-split: is-volatile r =
  (is-volatile-Read$_{sb}$ r ∨ is-volatile-Write$_{sb}$ r)
  by (cases r) auto

lemma is-non-volatile-split:
  ¬ is-volatile r = (is-non-volatile-Read$_{sb}$ r ∨ is-non-volatile-Write$_{sb}$ r ∨ is-Prog$_{sb}$ r ∨ is-Ghost$_{sb}$ r)
  by (cases r) auto

fun outstanding-refs:: (′p memref ⇒ bool) ⇒ ′p memref list ⇒ addr set
where
  outstanding-refs P [] = {}
| outstanding-refs P (r#rs) = (if P r then (address-of r) ∪ (outstanding-refs P rs)
                              else outstanding-refs P rs)

lemma outstanding-refs-conv: outstanding-refs P sb = ⋃(address-of ‘ {r. r ∈ set sb ∧ P r})
  by (induct sb) auto

lemma outstanding-refs-append:
  ⋀ys. outstanding-refs vol (xs@ys) = outstanding-refs vol xs ∪ outstanding-refs vol ys
  by (auto simp add: outstanding-refs-conv)

lemma outstanding-refs-empty-negate: (outstanding-refs P sb = {}) $\Longrightarrow$
(outstanding-refs (Not $\circ$ P) sb = $\bigcup$(address-of ` set sb))
by (auto simp add: outstanding-refs-conv)

lemma outstanding-refs-mono-pred:
$\bigwedge$sb sb$'$.
$\forall$r. P r $\longrightarrow$ P$'$ r $\Longrightarrow$ outstanding-refs P sb $\subseteq$ outstanding-refs P$'$ sb
by (auto simp add: outstanding-refs-conv)

lemma outstanding-refs-mono-set:
$\bigwedge$sb sb$'$.
set sb $\subseteq$ set sb$'$ $\Longrightarrow$ outstanding-refs P sb $\subseteq$ outstanding-refs P sb$'$
by (auto simp add: outstanding-refs-conv)

lemma outstanding-refs-takeWhile:
outstanding-refs P (takeWhile P$'$ sb) $\subseteq$ outstanding-refs P sb
apply (rule outstanding-refs-mono-set)
apply (auto dest: set-takeWhileD)
done

lemma outstanding-refs-subsets:
outstanding-refs is-volatile-Write$_{sb}$ sb $\subseteq$ outstanding-refs is-Write$_{sb}$ sb
outstanding-refs is-non-volatile-Write$_{sb}$ sb $\subseteq$ outstanding-refs is-Write$_{sb}$ sb

outstanding-refs is-volatile-Read$_{sb}$ sb $\subseteq$ outstanding-refs is-Read$_{sb}$ sb
outstanding-refs is-non-volatile-Read$_{sb}$ sb $\subseteq$ outstanding-refs is-Read$_{sb}$ sb

outstanding-refs is-non-volatile-Write$_{sb}$ sb $\subseteq$ outstanding-refs (Not $\circ$ is-volatile) sb
outstanding-refs is-non-volatile-Read$_{sb}$ sb $\subseteq$ outstanding-refs (Not $\circ$ is-volatile) sb

outstanding-refs is-volatile-Write$_{sb}$ sb $\subseteq$ outstanding-refs (is-volatile) sb
outstanding-refs is-volatile-Read$_{sb}$ sb $\subseteq$ outstanding-refs (is-volatile) sb

outstanding-refs is-non-volatile-Write$_{sb}$ sb $\subseteq$ outstanding-refs (Not $\circ$ is-volatile-Write$_{sb}$) sb
outstanding-refs is-non-volatile-Read$_{sb}$ sb $\subseteq$ outstanding-refs (Not $\circ$ is-volatile-Write$_{sb}$) sb
outstanding-refs is-volatile-Read$_{sb}$ sb $\subseteq$ outstanding-refs (Not $\circ$ is-volatile-Write$_{sb}$) sb
outstanding-refs is-Read$_{sb}$ sb $\subseteq$ outstanding-refs (Not $\circ$ is-volatile-Write$_{sb}$) sb
by (auto intro!:outstanding-refs-mono-pred simp add: is-volatile-Write$_{sb}$-def is-non-volatile-Write$_{sb}$-def
is-volatile-Read$_{sb}$-def is-non-volatile-Read$_{sb}$-def is-Read$_{sb}$-def split: memref.splits)

lemma outstanding-non-volatile-refs-conv:
outstanding-refs (Not $\circ$ is-volatile) sb =
outstanding-refs is-non-volatile-Write$_{sb}$ sb $\cup$ outstanding-refs is-non-volatile-Read$_{sb}$ sb
apply (induct sb)
apply simp

subgoal for a sb
by (case-tac a, auto)
done

lemma outstanding-volatile-refs-conv:
outstanding-refs is-volatile sb =
outstanding-refs is-volatile-Write$_{sb}$ sb ∪ outstanding-refs is-volatile-Read$_{sb}$ sb
apply (induct sb)
apply simp
subgoal for a sb
by (case-tac a, auto)
done

lemma outstanding-is-Write$_{sb}$-refs-conv:
outstanding-refs is-Write$_{sb}$ sb =
outstanding-refs is-non-volatile-Write$_{sb}$ sb ∪ outstanding-refs is-volatile-Write$_{sb}$ sb
apply (induct sb)
apply simp
subgoal for a sb
by (case-tac a, auto)
done

lemma outstanding-is-Read$_{sb}$-refs-conv:
outstanding-refs is-Read$_{sb}$ sb =
outstanding-refs is-non-volatile-Read$_{sb}$ sb ∪ outstanding-refs is-volatile-Read$_{sb}$ sb
apply (induct sb)
apply simp
subgoal for a sb
by (case-tac a, auto)
done

lemma outstanding-not-volatile-Read$_{sb}$-refs-conv: outstanding-refs (Not ∘ is-volatile-Read$_{sb}$) sb =
outstanding-refs is-Write$_{sb}$ sb ∪ outstanding-refs is-non-volatile-Read$_{sb}$ sb
apply (induct sb)
apply (clarsimp)
subgoal for a sb
by (case-tac a, auto)
done

lemmas misc-outstanding-refs-convs = outstanding-non-volatile-refs-conv outstanding-volatile-refs-conv
outstanding-is-Write$_{sb}$-refs-conv outstanding-is-Read$_{sb}$-refs-conv outstanding-not-volatile-Read$_{sb}$-refs-conv

lemma no-outstanding-vol-write-takeWhile-append: outstanding-refs is-volatile-Write$_{sb}$ sb = {} ⟹

takeWhile (Not ∘ is-volatile-Write$_{sb}$) (sb@xs) = sb@(takeWhile (Not ∘ is-volatile-Write$_{sb}$) xs)
apply (induct sb)
apply (auto split: if-split-asm)
done

lemma outstanding-vol-write-takeWhile-append: outstanding-refs is-volatile-Write$_{sb}$ sb ≠ {} ⟹
takeWhile (Not ∘ is-volatile-Write$_{sb}$) (sb@xs) = (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
apply (induct sb)
apply (auto split: if-split-asm)
done

lemma no-outstanding-vol-write-dropWhile-append: outstanding-refs is-volatile-Write$_{sb}$ sb = {} ⟹
dropWhile (Not ∘ is-volatile-Write$_{sb}$) (sb@xs) = (dropWhile (Not ∘ is-volatile-Write$_{sb}$) xs)
apply (induct sb)
apply (auto split: if-split-asm)
done

lemma outstanding-vol-write-dropWhile-append: outstanding-refs is-volatile-Write$_{sb}$ sb ≠ {} ⟹
dropWhile (Not ∘ is-volatile-Write$_{sb}$) (sb@xs) = (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb)@xs
apply (induct sb)
apply (auto split: if-split-asm)
done

lemmas outstanding-vol-write-take-drop-appends =
no-outstanding-vol-write-takeWhile-append
outstanding-vol-write-takeWhile-append
no-outstanding-vol-write-dropWhile-append
outstanding-vol-write-dropWhile-append

lemma outstanding-refs-is-non-volatile-Write$_{sb}$-takeWhile-conv:
outstanding-refs is-non-volatile-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) =
outstanding-refs is-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
apply (induct sb)
apply clarsimp
subgoal for a sb
by (case-tac a, auto)
done

lemma dropWhile-not-vol-write-empty:
outstanding-refs is-volatile-Write$_{sb}$ sb = {} ⟹ (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb) = []

apply (induct sb)
apply (auto split: if-split-asm)
done

lemma takeWhile-not-vol-write-outstanding-refs:
outstanding-refs is-volatile-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) = {}
apply (induct sb)
apply (auto split: if-split-asm)
done

lemma no-volatile-Write$_{sb}$s-conv: (outstanding-refs is-volatile-Write$_{sb}$ sb = {}) =
($\forall$ r $\in$ set sb. ($\forall$ v′ sop′ a′ A L R W. r $\neq$ Write$_{sb}$ True a′ sop′ v′ A L R W))
by (force simp add: outstanding-refs-conv is-volatile-Write$_{sb}$-def split: memref.splits)

lemma no-volatile-Read$_{sb}$s-conv: (outstanding-refs is-volatile-Read$_{sb}$ sb = {}) =
($\forall$ r $\in$ set sb. ($\forall$ v′ t′ a′. r $\neq$ Read$_{sb}$ True a′ t′ v′))
by (force simp add: outstanding-refs-conv is-volatile-Read$_{sb}$-def split: memref.splits)

inductive sb-memop-step :: (instrs × tmps × ′p store-buffer × memory × ′dirty × ′owns × ′rels × ′shared) ⇒
(instrs × tmps × ′p store-buffer × memory × ′dirty × ′owns × ′rels × ′shared) ⇒ bool
(‹- $\rightarrow_{sb}$ -› [60,60] 100)
where
SBReadBuffered:
⟦buffered-val sb a = Some v⟧
⟹
(Read volatile a t # is,j, sb, m,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{sb}$
(is, j (t↦v), sb, m,$\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$, $\mathcal{S}$)

| SBReadUnbuffered:
⟦buffered-val sb a = None⟧
⟹
(Read volatile a t # is, j, sb, m,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{sb}$
(is, j (t↦m a), sb, m,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$)

| SBWriteNonVolatile:
(Write False a (D,f) A L R W#is, j, sb, m,$\mathcal{D}$,$\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{sb}$
(is, j, sb@ [Write$_{sb}$ False a (D,f) (f j) A L R W], m,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$)

| SBWriteVolatile:

(Write True a (D,f) A L R W# is, j, sb, m,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{sb}$
(is, j, sb@[Write$_{sb}$ True a (D,f) (f j) A L R W], m,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$)

| SBFence:
(Fence # is, j, [], m,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{sb}$ (is, j, [], m,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$)

| SBRMWReadOnly:
⟦¬ cond (j(t↦m a))⟧ ⟹
(RMW a t (D,f) cond ret A L R W# is, j, [], m,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{\mathsf{sb}}$ (is, j(t↦m a),[], m,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$)

| SBRMWWrite:
⟦cond (j(t↦m a))⟧ ⟹
(RMW a t (D,f) cond ret A L R W# is, j, [], m,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{\mathsf{sb}}$
(is, j(t↦ret (m a) (f(j(t↦m a)))),[], m(a:= f(j(t↦m a))),$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$)

| SBGhost:
(Ghost A L R W# is, j, sb, m,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{\mathsf{sb}}$
(is, j, sb, m,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$)

inductive sbh-memop-step ::
(instrs × tmps × ′p store-buffer × memory × bool × owns × rels × shared) ⇒
(instrs × tmps × ′p store-buffer × memory × bool × owns × rels × shared) ⇒ bool
(‹- $\rightarrow_{\mathsf{sbh}}$ -› [60,60] 100)
where
SBHReadBuffered:
⟦buffered-val sb a = Some v⟧
⟹
(Read volatile a t # is, j, sb, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$
(is, j (t↦v), sb@[Read$_{\mathsf{sb}}$ volatile a t v], m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$)

| SBHReadUnbuffered:
⟦buffered-val sb a = None⟧
⟹
(Read volatile a t # is, j, sb, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$
(is, j (t↦m a), sb@[Read$_{\mathsf{sb}}$ volatile a t (m a)], m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$)

| SBHWriteNonVolatile:
(Write False a (D,f) A L R W#is, j, sb, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$
(is, j, sb@ [Write$_{\mathsf{sb}}$ False a (D,f) (f j) A L R W], m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$)

| SBHWriteVolatile:
(Write True a (D,f) A L R W# is, j, sb, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$
(is, j, sb@[Write$_{\mathsf{sb}}$ True a (D,f) (f j) A L R W], m, True, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$)

| SBHFence:
(Fence # is, j, [], m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$ (is, j, [], m, False, $\mathcal{O}$, Map.empty, $\mathcal{S}$)

| SBHRMWReadOnly:
⟦¬ cond (j(t↦m a))⟧ ⟹
(RMW a t (D,f) cond ret A L R W# is, j, [], m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$ (is, j(t↦m a),[], m, False, $\mathcal{O}$, Map.empty, $\mathcal{S}$)

| SBHRMWWrite:
⟦cond (j(t↦m a))⟧ ⟹
(RMW a t (D,f) cond ret A L R W# is, j, [], m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$
(is, j(t↦ret (m a) (f(j(t↦m a)))),[], m(a:= f(j(t↦m a))), False, $\mathcal{O}$ ∪ A − R,Map.empty, $\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)

| SBHGhost:
(Ghost A L R W# is, j, sb, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$
(is, j, sb@[Ghost$_{\mathsf{sb}}$ A L R W], m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$)

interpretation direct: memory-system direct-memop-step id-storebuffer-step **.**
interpretation sb: memory-system sb-memop-step store-buffer-step **.**
interpretation sbh: memory-system sbh-memop-step flush-step **.**

primrec non-volatile-owned-or-read-only:: bool ⇒ shared ⇒ owns ⇒ ′a memref list ⇒ bool
where
non-volatile-owned-or-read-only pending-write $\mathcal{S}$ $\mathcal{O}$ [] = True
| non-volatile-owned-or-read-only pending-write $\mathcal{S}$ $\mathcal{O}$ (x#xs) =
(case x of
Read$_{\mathsf{sb}}$ volatile a t v ⇒
(¬volatile ⟶ pending-write ⟶ (a ∈ $\mathcal{O}$ ∨ a ∈ read-only $\mathcal{S}$)) ∧
non-volatile-owned-or-read-only pending-write $\mathcal{S}$ $\mathcal{O}$ xs
| Write$_{\mathsf{sb}}$ volatile a sop v A L R W ⇒
(if volatile then non-volatile-owned-or-read-only True ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ ∪ A − R) xs
else a ∈ $\mathcal{O}$ ∧ non-volatile-owned-or-read-only pending-write $\mathcal{S}$ $\mathcal{O}$ xs)
| Ghost$_{\mathsf{sb}}$ A L R W ⇒ non-volatile-owned-or-read-only pending-write ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ ∪ A − R) xs
| - ⇒ non-volatile-owned-or-read-only pending-write $\mathcal{S}$ $\mathcal{O}$ xs)

primrec acquired :: bool ⇒ ′a memref list ⇒ addr set ⇒ addr set
where
acquired pending-write [] A = (if pending-write then A else {})
| acquired pending-write (x#xs) A =
(case x of
Write$_{\mathsf{sb}}$ volatile - - - A′ L R W⇒
(if volatile then acquired True xs (if pending-write then (A ∪ A′ − R) else (A′ − R))
else acquired pending-write xs A)
| Ghost$_{\mathsf{sb}}$ A′ L R W ⇒ acquired pending-write xs (if pending-write then (A ∪ A′ − R) else A)
| - ⇒ acquired pending-write xs A)

primrec share :: ′a memref list ⇒ shared ⇒ shared
where
share [] S = S
| share (x#xs) S =
(case x of

Write$_{sb}$ volatile - - - A L R W ⇒ (if volatile then (share xs (S $\oplus_W$ R $\ominus_A$ L)) else share xs S)
| Ghost$_{sb}$ A L R W ⇒ share xs (S $\oplus_W$ R $\ominus_A$ L)
| - ⇒ share xs S)

primrec acquired-reads :: bool ⇒ 'a memref list ⇒ addr set ⇒ addr set
where
acquired-reads pending-write [] A = {}
| acquired-reads pending-write (x#xs) A =
(case x of
Read$_{sb}$ volatile a t v ⇒ (if pending-write ∧ ¬ volatile ∧ a ∈ A
then insert a (acquired-reads pending-write xs A)
else acquired-reads pending-write xs A)
| Write$_{sb}$ volatile - - - A' L R W ⇒
(if volatile then acquired-reads True xs (if pending-write then (A ∪ A' − R) else (A' − R))
else acquired-reads pending-write xs A)
| Ghost$_{sb}$ A' L R W ⇒ acquired-reads pending-write xs (A ∪ A' − R)
| - ⇒ acquired-reads pending-write xs A)

lemma union-mono-aux: A ⊆ B ⟹ A ∪ C ⊆ B ∪ C
by blast

lemma set-minus-mono-aux: A ⊆ B ⟹ A − C ⊆ B − C
by blast

lemma acquired-mono: ⋀A B pending-write. A ⊆ B ⟹ acquired pending-write xs A ⊆ acquired pending-write xs B
apply (induct xs)
apply simp
subgoal for a xs A B pending-write
apply (case-tac a)
apply clarsimp
subgoal for volatile a1 D f v A' L R W x
apply (drule-tac C=A' **in** union-mono-aux)
apply (drule-tac C=R **in** set-minus-mono-aux)
apply blast
done
apply clarsimp
apply clarsimp
apply clarsimp
subgoal for A' L R W x
apply (drule-tac C=A' **in** union-mono-aux)
apply (drule-tac C=R **in** set-minus-mono-aux)
apply blast
done
done
done

lemma acquired-mono-in:
  assumes x-in: $x \in$ acquired pending-write xs A
  assumes sub: $A \subseteq B$
  shows $x \in$ acquired pending-write xs B
using acquired-mono [OF sub, of pending-write xs] x-in
by blast

lemma acquired-no-pending-write:$\bigwedge$A B. acquired False xs A = acquired False xs B
  by (induct xs) (auto split: memref.splits)

lemma acquired-no-pending-write-in:
  $x \in$ acquired False xs A $\Longrightarrow$ $x \in$ acquired False xs B
  apply (subst acquired-no-pending-write)
  apply auto
  done

lemma acquired-pending-write-mono-in: $\bigwedge$A B. $x \in$ acquired False xs A $\Longrightarrow$ $x \in$ acquired True xs B
apply (induct xs)
apply (auto split: memref.splits if-split-asm intro: acquired-mono-in)
done

lemma acquired-pending-write-mono: acquired False xs A $\subseteq$ acquired True xs B
  by (auto intro: acquired-pending-write-mono-in)

lemma acquired-append: $\bigwedge$A pending-write. acquired pending-write (xs@ys) A =
 acquired (pending-write $\vee$ outstanding-refs is-volatile-Write$_{sb}$ xs $\neq$ {}) ys (acquired pending-write xs A)
  apply (induct xs)
  apply (auto split: memref.splits intro: acquired-no-pending-write-in)
  done

lemma acquired-take-drop:
  acquired (pending-write $\vee$ outstanding-refs is-volatile-Write$_{sb}$ (takeWhile P xs) $\neq$ {})
     (dropWhile P xs) (acquired pending-write (takeWhile P xs) A) =
   acquired pending-write xs A
proof –
    have acquired pending-write xs A = acquired pending-write ((takeWhile P xs)@(dropWhile P xs)) A
    by simp
  also
  from acquired-append [**where** xs=(takeWhile P xs) **and** ys=(dropWhile P xs)]
  have ... = acquired (pending-write $\vee$ outstanding-refs is-volatile-Write$_{sb}$ (takeWhile P xs) $\neq$ {})
     (dropWhile P xs) (acquired pending-write (takeWhile P xs) A)
    by simp
  finally show ?thesis
    by simp
qed

lemma share-mono: $\bigwedge$A B. dom A $\subseteq$ dom B $\Longrightarrow$ dom (share xs A) $\subseteq$ dom (share xs B)
apply (induct xs)
apply simp
subgoal for a xs A B
apply (case-tac a)
apply (clarsimp iff del: domIff)
    subgoal for volatile a1 D f v A′ L R W x
    apply (drule-tac C=R **and** x=W **in** augment-mono-aux)
    apply (drule-tac C=L **in** restrict-mono-aux)
    apply blast
    done
apply clarsimp
apply clarsimp
apply (clarsimp iff del: domIff)
subgoal for A′ L R W x
apply (drule-tac C=R **and** x=W **in** augment-mono-aux)
apply (drule-tac C=L **in** restrict-mono-aux)
apply blast
done
done
done

lemma share-mono-in:
  assumes x-in: x $\in$ dom (share xs A)
  assumes sub: dom A $\subseteq$ dom B
  shows x $\in$ dom (share xs B)
using share-mono [OF sub, of xs] x-in
by blast

lemma acquired-reads-mono:
  $\bigwedge$A B pending-write. A $\subseteq$ B $\Longrightarrow$ acquired-reads pending-write xs A $\subseteq$ acquired-reads pending-write xs B
apply (induct xs)
apply simp
subgoal for a xs A B pending-write
apply (case-tac a)
apply clarsimp
    subgoal for volatile a1 D f v A′ L R W x
    apply (drule-tac C=A′ **in** union-mono-aux)
    apply (drule-tac C=R **in** set-minus-mono-aux)
    apply blast
    done
apply clarsimp
apply blast
apply clarsimp
apply clarsimp
subgoal for A′ L R W x
apply (drule-tac C=A′ **in** union-mono-aux)
apply (drule-tac C=R **in** set-minus-mono-aux)
apply blast

done
done
done

lemma acquired-reads-mono-in:
  assumes x-in: x ∈ acquired-reads pending-write xs A
  assumes sub: A ⊆ B
  shows x ∈ acquired-reads pending-write xs B
using acquired-reads-mono [OF sub, of pending-write xs] x-in
by blast

lemma acquired-reads-no-pending-write: ⋀A B. acquired-reads False xs A = acquired-reads False xs B
  by (induct xs) (auto split: memref.splits)

lemma acquired-reads-no-pending-write-in:
x ∈ acquired-reads False xs A ⟹ x ∈ acquired-reads False xs B
  apply (subst acquired-reads-no-pending-write)
  apply blast
  done

lemma acquired-reads-pending-write-mono:
  ⋀A. acquired-reads False xs A ⊆ acquired-reads True xs A
  by (induct xs) (auto split: memref.splits intro: acquired-reads-mono-in)

lemma acquired-reads-pending-write-mono-in:
  assumes x-in: x ∈ acquired-reads False xs A
  shows x ∈ acquired-reads True xs A
using acquired-reads-pending-write-mono [of xs A] x-in
by blast

lemma acquired-reads-append: ⋀pending-write A. acquired-reads pending-write (xs@ys) A =
  acquired-reads pending-write xs A ∪
  acquired-reads (pending-write ∨ (outstanding-refs is-volatile-Write$_{sb}$ xs ≠ {})) ys
   (acquired pending-write xs A)
proof (induct xs)
  case Nil **thus** ?case **by** (auto dest: acquired-reads-no-pending-write-in)
next
  case (Cons x xs)
  show ?case
  proof (cases x)
    case (Write$_{sb}$ volatile a sop v A L R W)
    show ?thesis
    proof (cases volatile)
      case False
      show ?thesis
 using Cons.hyps
 by (auto simp add: Write$_{sb}$ False)
    next

case True
show ?thesis
using Cons.hyps
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ True)
qed
next
case ($\text{Read}_{\mathsf{sb}}$ volatile a t v)
show ?thesis
proof (cases volatile)
case False
show ?thesis
using Cons.hyps
by (auto simp add: $\text{Read}_{\mathsf{sb}}$ False)
next
case True
show ?thesis
using Cons.hyps
by (auto simp add: $\text{Read}_{\mathsf{sb}}$ True)
qed
next
case $\text{Prog}_{\mathsf{sb}}$
with Cons.hyps **show** ?thesis **by** auto
next
case ($\text{Ghost}_{\mathsf{sb}}$ A′ L R W)
have (acquired False xs (A ∪ A′ −R)) = (acquired False xs A)
by (simp add: acquired-no-pending-write)
with Cons.hyps **show** ?thesis **by** (auto simp add: $\text{Ghost}_{\mathsf{sb}}$)
qed
qed

lemma in-acquired-reads-no-pending-write-outstanding-write:
$\bigwedge$A. a ∈ acquired-reads False xs A $\Longrightarrow$ outstanding-refs (is-volatile-$\text{Write}_{\mathsf{sb}}$) xs ≠ {}
apply (induct xs)
apply simp
apply (auto split: memref.splits)
apply auto
done

lemma augment-read-only-mono: read-only $\mathcal{S}$ ⊆ read-only $\mathcal{S}'$ $\Longrightarrow$
read-only ($\mathcal{S} \oplus_{\mathsf{W}}$ R) ⊆ read-only ($\mathcal{S}' \oplus_{\mathsf{W}}$ R)
by (auto simp add: augment-shared-def read-only-def)

lemma restrict-read-only-mono: read-only $\mathcal{S}$ ⊆ read-only $\mathcal{S}'$ $\Longrightarrow$
read-only ($\mathcal{S} \ominus_{\mathsf{A}}$ L) ⊆ read-only ($\mathcal{S}' \ominus_{\mathsf{A}}$ L)
apply (clarsimp simp add: restrict-shared-def read-only-def split: option.splits if-split-asm)
apply (rule conjI)
apply blast
apply fastforce
done

lemma share-read-only-mono: $\bigwedge\mathcal{S}\ \mathcal{S}'$. read-only $\mathcal{S} \subseteq$ read-only $\mathcal{S}' \Longrightarrow$
read-only (share sb $\mathcal{S}$) $\subseteq$ read-only (share sb $\mathcal{S}'$)
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\mathsf{Write}_{\mathsf{sb}}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)
case False
with Cons $\mathsf{Write}_{\mathsf{sb}}$ **show** ?thesis **by** auto
next
case True
note ‹read-only $\mathcal{S} \subseteq$ read-only $\mathcal{S}'$›
from augment-read-only-mono [OF this]
have read-only ($\mathcal{S} \oplus_{\mathsf{W}}$ R) $\subseteq$ read-only ($\mathcal{S}' \oplus_{\mathsf{W}}$ R).
from restrict-read-only-mono [OF this, of A L]
have read-only ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $\subseteq$ read-only ($\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L).
from Cons.hyps [OF this]
show ?thesis
by (clarsimp simp add: $\mathsf{Write}_{\mathsf{sb}}$ True)
qed
next
case $\mathsf{Read}_{\mathsf{sb}}$ **with** Cons **show** ?thesis
by auto
next
case $\mathsf{Prog}_{\mathsf{sb}}$ **with** Cons **show** ?thesis
by auto
next
case ($\mathsf{Ghost}_{\mathsf{sb}}$ A L R W)
note ‹read-only $\mathcal{S} \subseteq$ read-only $\mathcal{S}'$›
from augment-read-only-mono [OF this]
have read-only ($\mathcal{S} \oplus_{\mathsf{W}}$ R) $\subseteq$ read-only ($\mathcal{S}' \oplus_{\mathsf{W}}$ R).
from restrict-read-only-mono [OF this, of A L]
have read-only ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $\subseteq$ read-only ($\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L).
from Cons.hyps [OF this]
show ?thesis
by (clarsimp simp add: $\mathsf{Ghost}_{\mathsf{sb}}$)
qed
qed

lemma non-volatile-owned-or-read-only-append:
$\bigwedge\mathcal{O}\ \mathcal{S}$ pending-write. non-volatile-owned-or-read-only pending-write $\mathcal{S}\ \mathcal{O}$ (xs@ys)
= (non-volatile-owned-or-read-only pending-write $\mathcal{S}\ \mathcal{O}$ xs $\wedge$

non-volatile-owned-or-read-only (pending-write $\vee$ outstanding-refs is-volatile-Write$_{\mathsf{sb}}$ xs $\neq$ {})
(share xs $\mathcal{S}$) (acquired True xs $\mathcal{O}$) ys)
apply (induct xs)
apply (auto split: memref.splits)
done

lemma non-volatile-owned-or-read-only-mono:
$\bigwedge\mathcal{O}$ $\mathcal{O}'$ $\mathcal{S}$ pending-write. $\mathcal{O} \subseteq \mathcal{O}' \Longrightarrow$ non-volatile-owned-or-read-only pending-write $\mathcal{S}$ $\mathcal{O}$ xs
$\Longrightarrow$ non-volatile-owned-or-read-only pending-write $\mathcal{S}$ $\mathcal{O}'$ xs
apply (induct xs)
apply simp
subgoal for a xs $\mathcal{O}$ $\mathcal{O}'$ $\mathcal{S}$ pending-write
apply (case-tac a)
apply (clarsimp split: if-split-asm)
subgoal for volatile a1 D f v A L R W
apply (drule-tac C=A **in** union-mono-aux)
apply (drule-tac C=R **in** set-minus-mono-aux)
apply blast
done
apply fastforce
apply fastforce
apply fastforce
apply clarsimp
subgoal for A L R W
apply (drule-tac C=A **in** union-mono-aux)
apply (drule-tac C=R **in** set-minus-mono-aux)
apply blast
done
done
done

lemma non-volatile-owned-or-read-only-shared-mono:
$\bigwedge\mathcal{S}$ $\mathcal{S}'$ $\mathcal{O}$ pending-write. $\mathcal{S} \subseteq_{\mathsf{s}} \mathcal{S}' \Longrightarrow$ non-volatile-owned-or-read-only pending-write $\mathcal{S}$ $\mathcal{O}$ xs
$\Longrightarrow$ non-volatile-owned-or-read-only pending-write $\mathcal{S}'$ $\mathcal{O}$ xs
apply (induct xs)
apply simp
subgoal for a xs $\mathcal{S}$ $\mathcal{S}'$ $\mathcal{O}$ pending-write
apply (case-tac a)
apply (clarsimp split: if-split-asm)
subgoal for volatile a1 D f v A L R W
apply (frule-tac C=R **and** x=W **in** augment-mono-map)
apply (drule-tac A=$\mathcal{S} \oplus_{\mathsf{W}}$ R **and** C=L **in** restrict-mono-map)
apply (fastforce dest: read-only-mono)
done
apply (fastforce dest: read-only-mono shared-leD)
apply fastforce
subgoal for A L R W

apply (frule-tac C=R **and** x=W **in** augment-mono-map)
apply (drule-tac A=$\mathcal{S} \oplus_{\mathsf{W}}$ R **and** C=L **in** restrict-mono-map)
apply (fastforce dest: read-only-mono)
done
done
done

lemma non-volatile-owned-or-read-only-pending-write-antimono:
$\bigwedge \mathcal{O}\ \mathcal{S}$. non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ xs
$\Longrightarrow$ non-volatile-owned-or-read-only False $\mathcal{S}$ $\mathcal{O}$ xs
by (induct xs) (auto split: memref.splits)

primrec all-acquired :: $'$a memref list $\Rightarrow$ addr set
where
all-acquired [] = {}
| all-acquired (i#is) =
(case i of
Write$_{\mathsf{sb}}$ volatile - - - A L R W $\Rightarrow$ (if volatile then A $\cup$ all-acquired is else all-acquired is)
| Ghost$_{\mathsf{sb}}$ A L R W $\Rightarrow$ A $\cup$ all-acquired is
| - $\Rightarrow$ all-acquired is)

lemma all-acquired-append: all-acquired (xs@ys) = all-acquired xs $\cup$ all-acquired ys
apply (induct xs)
apply (auto split: memref.splits)
done

lemma acquired-reads-all-acquired: $\bigwedge \mathcal{O}$ pending-write.
acquired-reads pending-write sb $\mathcal{O} \subseteq \mathcal{O} \cup$ all-acquired sb
apply (induct sb)
apply clarsimp
apply (auto split: memref.splits)
done

lemma acquired-takeWhile-non-volatile-Write$_{\mathsf{sb}}$:
$\bigwedge$A. (acquired True (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) A) $\subseteq$
A $\cup$ all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb)
apply (induct sb)
apply clarsimp
subgoal for a sb A
apply (case-tac a)
apply auto
done
done

lemma acquired-False-takeWhile-non-volatile-Write$_{\mathsf{sb}}$:
acquired False (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) A = {}
apply (induct sb)
apply simp

subgoal for a sb
  by (case-tac a) auto
done

lemma outstanding-refs-takeWhile-opposite: outstanding-refs P (takeWhile (Not ∘ P) xs) = {}
apply (induct xs)
apply auto
done

lemma no-outstanding-volatile-Write$_{sb}$-acquired:
outstanding-refs is-volatile-Write$_{sb}$ sb = {} $\Longrightarrow$ acquired False sb A = {}
apply (induct sb)
 apply simp
subgoal for a sb
  by (case-tac a) auto
done

lemma acquired-all-acquired: $\bigwedge$pending-write A. acquired pending-write xs A $\subseteq$ A $\cup$ all-acquired xs
apply (induct xs)
apply (auto split: memref.splits)
done

lemma acquired-all-acquired-in: x $\in$ acquired pending-write xs A $\Longrightarrow$ x $\in$ A $\cup$ all-acquired xs
using acquired-all-acquired
by blast

primrec sharing-consistent:: shared $\Rightarrow$ owns $\Rightarrow$ 'a memref list $\Rightarrow$ bool
where
sharing-consistent $\mathcal{S}$ $\mathcal{O}$ [] = True
| sharing-consistent $\mathcal{S}$ $\mathcal{O}$ (r#rs) =
 (case r of
  Write$_{sb}$ volatile - - - A L R W $\Rightarrow$
   (if volatile then A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ $\wedge$ L $\subseteq$ A $\wedge$ A $\cap$ R = {} $\wedge$ R $\subseteq$ $\mathcal{O}$ $\wedge$
        sharing-consistent ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ $\cup$ A $-$ R) rs
   else sharing-consistent $\mathcal{S}$ $\mathcal{O}$ rs)
 | Ghost$_{sb}$ A L R W $\Rightarrow$ A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ $\wedge$ L $\subseteq$ A $\wedge$ A $\cap$ R = {} $\wedge$ R $\subseteq$ $\mathcal{O}$ $\wedge$
    sharing-consistent ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ $\cup$ A $-$ R) rs
 | - $\Rightarrow$ sharing-consistent $\mathcal{S}$ $\mathcal{O}$ rs)

lemma sharing-consistent-all-acquired:
$\bigwedge\mathcal{S}$ $\mathcal{O}$. sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb $\Longrightarrow$ all-acquired sb $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$
proof (induct sb)
case Nil **thus** ?case **by** simp
next

case (Cons x sb)
show ?case
proof (cases x)
case ($Write_{sb}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)
case False
with Cons $Write_{sb}$ **show** ?thesis **by** auto
next
case True
from Cons.hyps [**where** $\mathcal{S}$=($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) **and** $\mathcal{O}$=($\mathcal{O}$ ∪ A − R)] Cons.prems
show ?thesis
by (auto simp add: $Write_{sb}$ True)
qed
next
case $Read_{sb}$ **with** Cons **show** ?thesis **by** auto
next
case $Prog_{sb}$ **with** Cons **show** ?thesis **by** auto
next
case ($Ghost_{sb}$ A L R W)
with Cons.hyps [**where** $\mathcal{S}$=($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) **and** $\mathcal{O}$=($\mathcal{O}$ ∪ A − R)] Cons.prems **show** ?thesis **by** auto
qed
qed

lemma sharing-consistent-append:
⋀$\mathcal{S}$ $\mathcal{O}$. sharing-consistent $\mathcal{S}$ $\mathcal{O}$ (xs@ys) =
(sharing-consistent $\mathcal{S}$ $\mathcal{O}$ xs ∧
sharing-consistent (share xs $\mathcal{S}$) (acquired True xs $\mathcal{O}$) ys)
apply (induct xs)
apply (auto split: memref.splits)
done

primrec read-only-reads :: owns ⇒ ′a memref list ⇒ addr set
where
read-only-reads $\mathcal{O}$ [] = {}
| read-only-reads $\mathcal{O}$ (x#xs) =
(case x of
$Read_{sb}$ volatile a t v ⇒ (if ¬ volatile ∧ a ∉ $\mathcal{O}$
then insert a (read-only-reads $\mathcal{O}$ xs)
else read-only-reads $\mathcal{O}$ xs)
| $Write_{sb}$ volatile - - - A L R W ⇒
(if volatile then read-only-reads ($\mathcal{O}$ ∪ A − R) xs
else read-only-reads $\mathcal{O}$ xs)
| $Ghost_{sb}$ A L R W ⇒ read-only-reads ($\mathcal{O}$ ∪ A − R) xs
| - ⇒ read-only-reads $\mathcal{O}$ xs)

lemma read-only-reads-append:
⋀$\mathcal{O}$. read-only-reads $\mathcal{O}$ (xs@ys) =
read-only-reads $\mathcal{O}$ xs ∪ read-only-reads (acquired True xs $\mathcal{O}$) ys

apply (induct xs)
 apply simp
subgoal for a xs $\mathcal{O}$
 by (case-tac a) auto
done

lemma read-only-reads-antimono:
$\bigwedge \mathcal{O}\ \mathcal{O}'.$
$\mathcal{O} \subseteq \mathcal{O}' \Longrightarrow$ read-only-reads $\mathcal{O}'$ sb $\subseteq$ read-only-reads $\mathcal{O}$ sb
apply (induct sb)
apply simp
subgoal for a sb $\mathcal{O}$ $\mathcal{O}'$
apply (case-tac a)
apply (clarsimp split: if-split-asm)
subgoal for volatile a1 D f v A L R W
apply (drule-tac C=A **in** union-mono-aux)
apply (drule-tac C=R **in** set-minus-mono-aux)
apply blast
done
apply auto
subgoal for A L R W x
apply (drule-tac C=A **in** union-mono-aux)
apply (drule-tac C=R **in** set-minus-mono-aux)
apply blast
done
done
done

primrec non-volatile-writes-unshared:: shared $\Rightarrow$ $'$a memref list $\Rightarrow$ bool
where
non-volatile-writes-unshared $\mathcal{S}$ [] = True
| non-volatile-writes-unshared $\mathcal{S}$ (x#xs) =
(case x of
Write$_{\mathsf{sb}}$ volatile a sop v A L R W $\Rightarrow$ (if volatile then non-volatile-writes-unshared ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) xs
else a $\notin$ dom $\mathcal{S}$ $\wedge$ non-volatile-writes-unshared $\mathcal{S}$ xs)
| Ghost$_{\mathsf{sb}}$ A L R W $\Rightarrow$ non-volatile-writes-unshared ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) xs
| - $\Rightarrow$ non-volatile-writes-unshared $\mathcal{S}$ xs)

lemma non-volatile-writes-unshared-append:
$\bigwedge \mathcal{S}.$ non-volatile-writes-unshared $\mathcal{S}$ (xs@ys)
= (non-volatile-writes-unshared $\mathcal{S}$ xs $\wedge$ non-volatile-writes-unshared (share xs $\mathcal{S}$) ys)
apply (induct xs)
apply (auto split: memref.splits)
done

lemma non-volatile-writes-unshared-antimono:
$\bigwedge \mathcal{S}\ \mathcal{S}'.$ dom $\mathcal{S} \subseteq$ dom $\mathcal{S}' \Longrightarrow$ non-volatile-writes-unshared $\mathcal{S}'$ xs

$\Longrightarrow$ non-volatile-writes-unshared $\mathcal{S}$ xs
apply (induct xs)
apply simp
subgoal for a xs $\mathcal{S}$ $\mathcal{S}'$
apply (case-tac a)
apply (clarsimp split: if-split-asm)
subgoal for volatile a1 D f v A L R W
apply (drule-tac C=R **in** augment-mono-aux)
apply (drule-tac C=L **in** restrict-mono-aux)
apply blast
done
apply fastforce
apply fastforce
apply fastforce
apply (clarsimp split: if-split-asm)
subgoal for A L R W
apply (drule-tac C=R **in** augment-mono-aux)
apply (drule-tac C=L **in** restrict-mono-aux)
apply blast
done
done
done

primrec no-write-to-read-only-memory:: shared $\Rightarrow$ $'$a memref list $\Rightarrow$ bool
where
no-write-to-read-only-memory $\mathcal{S}$ [] = True
| no-write-to-read-only-memory $\mathcal{S}$ (x#xs) =
(case x of
Write$_{\mathsf{sb}}$ volatile a sop v A L R W $\Rightarrow$ a $\notin$ read-only $\mathcal{S}$ $\wedge$
(if volatile then no-write-to-read-only-memory ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) xs
else no-write-to-read-only-memory $\mathcal{S}$ xs)
| Ghost$_{\mathsf{sb}}$ A L R W $\Rightarrow$ no-write-to-read-only-memory ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) xs
| - $\Rightarrow$ no-write-to-read-only-memory $\mathcal{S}$ xs)

lemma no-write-to-read-only-memory-append:
$\bigwedge\mathcal{S}$. no-write-to-read-only-memory $\mathcal{S}$ (xs@ys)
= (no-write-to-read-only-memory $\mathcal{S}$ xs $\wedge$ no-write-to-read-only-memory (share xs $\mathcal{S}$) ys)
apply (induct xs)
apply simp
subgoal for a xs $\mathcal{S}$
by (case-tac a) auto
done

lemma no-write-to-read-only-memory-antimono:
$\bigwedge\mathcal{S}$ $\mathcal{S}'$. $\mathcal{S}$ $\subseteq_{\mathsf{s}}$ $\mathcal{S}'$ $\Longrightarrow$ no-write-to-read-only-memory $\mathcal{S}'$ xs
$\Longrightarrow$ no-write-to-read-only-memory $\mathcal{S}$ xs
apply (induct xs)
apply simp

subgoal for a xs $\mathcal{S}$ $\mathcal{S}'$
apply (case-tac a)
apply (clarsimp split: if-split-asm)
subgoal for volatile a1 D f v A L R W
apply (frule-tac C=R **and** x=W **in** augment-mono-map)
apply (drule-tac A=$\mathcal{S} \oplus_{\mathsf{W}}$ R **and** C=L **and** x=A **in** restrict-mono-map)
apply (fastforce dest: read-only-mono shared-leD)
done
apply (fastforce dest: read-only-mono shared-leD)
apply fastforce
apply fastforce
apply (clarsimp)
subgoal for A L R W
apply (frule-tac C=R **and** x=W **in** augment-mono-map)
apply (drule-tac A=$\mathcal{S} \oplus_{\mathsf{W}}$ R **and** C=L **and** x=A **in** restrict-mono-map)
apply (fastforce dest: read-only-mono shared-leD)
done
done
done

locale outstanding-non-volatile-refs-owned-or-read-only =
fixes $\mathcal{S}$::shared
fixes ts::('p,'p store-buffer,bool,owns,rels) thread-config list
assumes outstanding-non-volatile-refs-owned-or-read-only:
$\bigwedge$i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p.
$[\![$i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $]\!]$
$\Longrightarrow$
non-volatile-owned-or-read-only False $\mathcal{S}$ $\mathcal{O}$ sb

locale outstanding-volatile-writes-unowned-by-others =
fixes ts::('p,'p store-buffer,bool,owns,rels) thread-config list
assumes outstanding-volatile-writes-unowned-by-others:
$\bigwedge$i p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j_i sb_i j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j.
$[\![$i < length ts; j < length ts; i$\neq$j;
ts!i = (p_i,is_i,j_i,sb_i,$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$); ts!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
$]\!]$
$\Longrightarrow$
($\mathcal{O}_j$ $\cup$ all-acquired sb_j) $\cap$ outstanding-refs is-volatile-$\mathrm{Write}_{\mathsf{sb}}$ sb_i = {}

locale read-only-reads-unowned =
fixes ts::('p,'p store-buffer,bool,owns,rels) thread-config list
assumes read-only-reads-unowned:
$\bigwedge$i p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j_i sb_i j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j.
$[\![$i < length ts; j < length ts; i$\neq$j;
ts!i = (p_i,is_i,j_i,sb_i,$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$); ts!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
$]\!]$
$\Longrightarrow$
($\mathcal{O}_j$ $\cup$ all-acquired sb_j) $\cap$
read-only-reads (acquired True

(takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_i$) $\mathcal{O}_i$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_i$) = {}

locale ownership-distinct =
fixes ts::('p,'p store-buffer,bool,owns,rels) thread-config list
assumes ownership-distinct:
$\bigwedge$i j p$_i$ is$_i$ $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j$_i$ sb$_i$ p$_j$ is$_j$ $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j$_j$ sb$_j$.
⟦i < length ts; j < length ts; i ≠ j;
ts!i = (p$_i$,is$_i$,j$_i$,sb$_i$,$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$); ts!j = (p$_j$,is$_j$,j$_j$,sb$_j$,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
⟧ ⟹ ($\mathcal{O}_i$ ∪ all-acquired sb$_i$) ∩ ($\mathcal{O}_j$ ∪ all-acquired sb$_j$) = {}

locale valid-ownership =
outstanding-non-volatile-refs-owned-or-read-only +
outstanding-volatile-writes-unowned-by-others +
read-only-reads-unowned +
ownership-distinct

locale outstanding-non-volatile-writes-unshared =
fixes $\mathcal{S}$::shared **and** ts::('p,'p store-buffer,bool,owns,rels) thread-config list
assumes outstanding-non-volatile-writes-unshared:
$\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟧
⟹
non-volatile-writes-unshared $\mathcal{S}$ sb

locale sharing-consis =
fixes $\mathcal{S}$::shared **and** ts::('p,'p store-buffer,bool,owns,rels) thread-config list
assumes sharing-consis:
$\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟧
⟹
sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb

locale no-outstanding-write-to-read-only-memory =
fixes $\mathcal{S}$::shared **and** ts::('p,'p store-buffer,bool,owns,rels) thread-config list
assumes no-outstanding-write-to-read-only-memory:
$\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟧
⟹
no-write-to-read-only-memory $\mathcal{S}$ sb

locale valid-sharing =
outstanding-non-volatile-writes-unshared +
sharing-consis +

read-only-unowned +
unowned-shared +
no-outstanding-write-to-read-only-memory

locale valid-ownership-and-sharing = valid-ownership +
outstanding-non-volatile-writes-unshared +
sharing-consis +
no-outstanding-write-to-read-only-memory

lemma (**in** read-only-reads-unowned)
read-only-reads-unowned-nth-update:
$\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
$[\![$i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$'$) $\mathcal{O}'$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb$'$) ⊆ read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{O}$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb);
$\mathcal{O}'$ ∪ all-acquired sb$'$ ⊆ $\mathcal{O}$ ∪ all-acquired sb$]\!]$ $\Longrightarrow$
read-only-reads-unowned (ts[i := (p$'$,is$'$,j$'$,sb$'$,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)])
apply (unfold-locales)
apply (clarsimp simp add: nth-list-update split: if-split-asm)
apply (fastforce dest: read-only-reads-unowned)+
done

lemma outstanding-non-volatile-refs-owned-or-read-only-tl:
outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}$ (t#ts) $\Longrightarrow$ outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}$ ts
by (force simp add: outstanding-non-volatile-refs-owned-or-read-only-def)

lemma outstanding-volatile-writes-unowned-by-others-tl:
outstanding-volatile-writes-unowned-by-others (t#ts) $\Longrightarrow$ outstanding-volatile-writes-unowned-by-others ts
apply (clarsimp simp add: outstanding-volatile-writes-unowned-by-others-def)
apply fastforce
done

lemma read-only-reads-unowned-tl:
read-only-reads-unowned (t # ts) $\Longrightarrow$
read-only-reads-unowned (ts)
apply (clarsimp simp add: read-only-reads-unowned-def)
apply fastforce
done

lemma ownership-distinct-tl:
assumes dist: ownership-distinct (t#ts)

shows ownership-distinct ts
proof −
from dist
interpret ownership-distinct t#ts **.**

show ?thesis
proof (rule ownership-distinct.intro)
fix i j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ xs sb p′ is′ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{D}'$ xs′ sb′
assume i-bound: i < length ts
and j-bound: j < length ts
and neq: i ≠ j
and ith: ts ! i = (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
and jth: ts ! j = (p′,is′, xs′, sb′,$\mathcal{D}'$, $\mathcal{O}'$,$\mathcal{R}'$)
from i-bound j-bound neq ith jth
show ($\mathcal{O}$ ∪ all-acquired sb) ∩ ($\mathcal{O}'$ ∪ all-acquired sb′) = {}
by − (rule ownership-distinct [of Suc i Suc j],auto)
qed
qed

lemma valid-ownership-tl: valid-ownership $\mathcal{S}$ (t#ts) ⟹ valid-ownership $\mathcal{S}$ ts
by (auto simp add: valid-ownership-def
intro: outstanding-volatile-writes-unowned-by-others-tl
outstanding-non-volatile-refs-owned-or-read-only-tl ownership-distinct-tl
read-only-reads-unowned-tl)

lemma sharing-consistent-takeWhile:
assumes consis: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb
shows sharing-consistent $\mathcal{S}$ $\mathcal{O}$ (takeWhile P sb)
proof −
from consis **have** sharing-consistent $\mathcal{S}$ $\mathcal{O}$ (takeWhile P sb @ dropWhile P sb)
by simp
with sharing-consistent-append [of - - takeWhile P sb dropWhile P sb]
show ?thesis
by simp
qed

lemma sharing-consis-tl: sharing-consis $\mathcal{S}$ (t#ts) ⟹ sharing-consis $\mathcal{S}$ ts
by (auto simp add: sharing-consis-def)

lemma sharing-consis-Cons:
⟦sharing-consis $\mathcal{S}$ ts; sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb⟧
⟹ sharing-consis $\mathcal{S}$ ((p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)#ts)
apply (clarsimp simp add: sharing-consis-def)
subgoal for i pa isa $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{D}'$ j′ sba
by (case-tac i) auto
done

lemma outstanding-non-volatile-writes-unshared-tl:
outstanding-non-volatile-writes-unshared $\mathcal{S}$ (t#ts) ⟹

outstanding-non-volatile-writes-unshared $\mathcal{S}$ ts
by (auto simp add: outstanding-non-volatile-writes-unshared-def)

lemma no-outstanding-write-to-read-only-memory-tl:
no-outstanding-write-to-read-only-memory $\mathcal{S}$ (t#ts) $\Longrightarrow$
no-outstanding-write-to-read-only-memory $\mathcal{S}$ ts
by (auto simp add: no-outstanding-write-to-read-only-memory-def)

lemma valid-ownership-and-sharing-tl:
valid-ownership-and-sharing $\mathcal{S}$ (t#ts) $\Longrightarrow$ valid-ownership-and-sharing $\mathcal{S}$ ts
apply (clarsimp simp add: valid-ownership-and-sharing-def)
apply (auto intro: valid-ownership-tl
outstanding-non-volatile-writes-unshared-tl
no-outstanding-write-to-read-only-memory-tl
sharing-consis-tl)
done

lemma non-volatile-owned-or-read-only-outstanding-non-volatile-writes:
$\bigwedge\mathcal{O}$ $\mathcal{S}$ pending-write. $[\![$non-volatile-owned-or-read-only pending-write $\mathcal{S}$ $\mathcal{O}$ sb$]\!]$
$\Longrightarrow$
outstanding-refs is-non-volatile-Write$_{\mathsf{sb}}$ sb $\subseteq$ $\mathcal{O}$ $\cup$ all-acquired sb
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case (Write$_{\mathsf{sb}}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)
case True
from Cons.hyps [of True ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ $\cup$ A $-$ R)] Cons.prems
show ?thesis
by (auto simp add: Write$_{\mathsf{sb}}$ True)
next
case False **with** Cons **show** ?thesis
by (auto simp add: Write$_{\mathsf{sb}}$)
qed
next
case Read$_{\mathsf{sb}}$ **with** Cons **show** ?thesis
by auto
next
case Prog$_{\mathsf{sb}}$ **with** Cons **show** ?thesis
by auto
next
case (Ghost$_{\mathsf{sb}}$ A L R W)
from Cons.hyps [of pending-write ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ $\cup$ A $-$ R)] Cons.prems
show ?thesis
by (auto simp add: Ghost$_{\mathsf{sb}}$)

qed
qed

lemma (**in** outstanding-non-volatile-refs-owned-or-read-only) outstanding-non-volatile-writes-owned:
assumes i-bound: i < length ts
assumes ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
shows outstanding-refs is-non-volatile-Write$_{sb}$ sb $\subseteq$ $\mathcal{O}$ $\cup$ all-acquired sb
using non-volatile-owned-or-read-only-outstanding-non-volatile-writes [OF outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ts-i]]
by blast

lemma non-volatile-reads-acquired-or-read-only:
$\bigwedge$$\mathcal{O}$ $\mathcal{S}$. ⟦non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb; sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb⟧
$\Longrightarrow$
outstanding-refs is-non-volatile-Read$_{sb}$ sb $\subseteq$ $\mathcal{O}$ $\cup$ all-acquired sb $\cup$ read-only $\mathcal{S}$
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case (Write$_{sb}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)
case True

from Cons.prems **obtain** non-vol: non-volatile-owned-or-read-only True ($\mathcal{S}$ $\oplus_{W}$ R $\ominus_{A}$ L) ($\mathcal{O}$ $\cup$ A $-$ R) sb **and**
A-shared-onws: A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and**
consis$'$: sharing-consistent ($\mathcal{S}$ $\oplus_{W}$ R $\ominus_{A}$ L) ($\mathcal{O}$ $\cup$ A $-$ R) sb
by (clarsimp simp add: Write$_{sb}$ True)

from Cons.hyps [OF non-vol consis$'$]
have hyp: outstanding-refs is-non-volatile-Read$_{sb}$ sb
$\subseteq$ $\mathcal{O}$ $\cup$ A $-$ R $\cup$ all-acquired sb $\cup$ read-only ($\mathcal{S}$ $\oplus_{W}$ R $\ominus_{A}$ L).
with R-owns A-R L-A
show ?thesis
apply (clarsimp simp add: Write$_{sb}$ True)
apply (drule (1) rev-subsetD)
apply (auto simp add: in-read-only-convs split: if-split-asm)
done
next

case False **with** Cons **show** ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$)
qed
next
case $\text{Read}_{\mathsf{sb}}$ **with** Cons **show** ?thesis
by auto
next
case $\text{Prog}_{\mathsf{sb}}$ **with** Cons **show** ?thesis
by auto
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.prems **obtain** non-vol: non-volatile-owned-or-read-only True ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb **and**
A-shared-onws: A $\subseteq$ dom $\mathcal{S} \cup \mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq \mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)

from Cons.hyps [OF non-vol consis′]
have hyp: outstanding-refs is-non-volatile-$\text{Read}_{\mathsf{sb}}$ sb
$\subseteq \mathcal{O} \cup$ A $-$ R $\cup$ all-acquired sb $\cup$ read-only ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L).
with R-owns A-R L-A
show ?thesis
apply (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
apply (drule (1) rev-subsetD)
apply (auto simp add: in-read-only-convs split: if-split-asm)
done
qed
qed

lemma non-volatile-reads-acquired-or-read-only-reads:
$\bigwedge\mathcal{O}$ $\mathcal{S}$ pending-write. [[non-volatile-owned-or-read-only pending-write $\mathcal{S}$ $\mathcal{O}$ sb]]
$\Longrightarrow$
outstanding-refs is-non-volatile-$\text{Read}_{\mathsf{sb}}$ sb $\subseteq \mathcal{O} \cup$ all-acquired sb $\cup$ read-only-reads $\mathcal{O}$ sb
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)
case True

from Cons.prems **obtain** non-vol: non-volatile-owned-or-read-only True ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ True)

from Cons.hyps [OF non-vol]
have hyp: outstanding-refs is-non-volatile-Read$_{sb}$ sb
$\subseteq \mathcal{O} \cup A - R \cup$ all-acquired sb $\cup$ read-only-reads ($\mathcal{O} \cup A - R$) sb**.**
then
show ?thesis
by (auto simp add: Write$_{sb}$ True)
next
case False **with** Cons **show** ?thesis
by (auto simp add: Write$_{sb}$)
qed
next
case Read$_{sb}$ **with** Cons **show** ?thesis
by auto
next
case Prog$_{sb}$ **with** Cons **show** ?thesis
by auto
next
case (Ghost$_{sb}$ A L R W)
from Cons.prems **obtain** non-vol: non-volatile-owned-or-read-only pending-write ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) ($\mathcal{O} \cup A - R$) sb
by (clarsimp simp add: Ghost$_{sb}$)

from Cons.hyps [OF non-vol]
have hyp: outstanding-refs is-non-volatile-Read$_{sb}$ sb
$\subseteq \mathcal{O} \cup A - R \cup$ all-acquired sb $\cup$ read-only-reads ($\mathcal{O} \cup A - R$) sb**.**
then
show ?thesis
by (auto simp add: Ghost$_{sb}$)
qed
qed

lemma non-volatile-owned-or-read-only-outstanding-refs:
$\bigwedge\mathcal{O}$ $\mathcal{S}$ pending-write. ⟦non-volatile-owned-or-read-only pending-write $\mathcal{S}$ $\mathcal{O}$ sb⟧
$\Longrightarrow$
outstanding-refs (Not $\circ$ is-volatile) sb $\subseteq \mathcal{O} \cup$ all-acquired sb $\cup$ read-only-reads $\mathcal{O}$ sb
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case (Write$_{sb}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)
case True
from Cons.hyps [of True ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) ($\mathcal{O} \cup A - R$)] Cons.prems
show ?thesis

by (auto simp add: $\text{Write}_{\mathsf{sb}}$ True)
  next
    case False **with** Cons **show** ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$)
  qed
 next
  case $\text{Read}_{\mathsf{sb}}$ **with** Cons **show** ?thesis
    by auto
 next
  case $\text{Prog}_{\mathsf{sb}}$ **with** Cons **show** ?thesis
    by auto
 next
  case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
  from Cons.hyps [of pending-write ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R)] Cons.prems
  show ?thesis
    by (auto simp add: $\text{Ghost}_{\mathsf{sb}}$)
 qed
qed

lemma no-unacquired-write-to-read-only:
$\bigwedge\mathcal{S}\ \mathcal{O}$. $[\![$no-write-to-read-only-memory $\mathcal{S}$ sb;sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb;
a $\in$ read-only $\mathcal{S}$; a $\notin$ ($\mathcal{O} \cup$ all-acquired sb)$]\!]$
  $\Longrightarrow$ a $\notin$ outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ sb
proof (induct sb)
 case Nil **thus** ?case **by** simp
next
 case (Cons x sb)
 show ?case
 proof (cases x)
  case ($\text{Write}_{\mathsf{sb}}$ volatile a′ sop v A L R W)
  show ?thesis
  proof (cases volatile)
    case True

    from Cons.prems **obtain** no-wrt: no-write-to-read-only-memory ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) sb
and
A-shared-onws: A $\subseteq$ dom $\mathcal{S} \cup \mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns:
R $\subseteq \mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb **and**
a-ro: a $\in$ read-only $\mathcal{S}$ **and**
a-A: a $\notin$ A **and** a-all-acq: a $\notin$ all-acquired sb **and** a-owns: a $\notin \mathcal{O}$ **and**
a′-notin: a′ $\notin$ read-only $\mathcal{S}$
by (simp add: $\text{Write}_{\mathsf{sb}}$ True)

    from a′-notin a-ro **have** neq-a-a′: a$\neq$a′
by blast

    from a-A a-all-acq a-owns

have a-notin′: a $\notin \mathcal{O} \cup$ A $-$ R $\cup$ all-acquired sb
by auto
from a-ro L-A a-A R-owns a-owns
have a $\in$ read-only ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (auto simp add: in-read-only-convs split: if-split-asm)

from Cons.hyps [OF no-wrt consis′ this a-notin′]
have a $\notin$ outstanding-refs is-Write$_{\mathsf{sb}}$ sb**.**
with neq-a-a′
show ?thesis
by (clarsimp simp add: Write$_{\mathsf{sb}}$ True)
next
case False **with** Cons
show ?thesis
by (auto simp add: Write$_{\mathsf{sb}}$ False)
qed
next
case Read$_{\mathsf{sb}}$ **with** Cons
show ?thesis
by (auto)
next
case Prog$_{\mathsf{sb}}$ **with** Cons
show ?thesis
by (auto)
next
case (Ghost$_{\mathsf{sb}}$ A L R W)
from Cons.prems **obtain** no-wrt: no-write-to-read-only-memory ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) sb **and**
A-shared-onws: A $\subseteq$ dom $\mathcal{S} \cup \mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R $=$ {} **and** R-owns: R $\subseteq \mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb **and**
a-ro: a $\in$ read-only $\mathcal{S}$ **and**
a-A: a $\notin$ A **and** a-all-acq: a $\notin$ all-acquired sb **and** a-owns: a $\notin \mathcal{O}$
by (simp add: Ghost$_{\mathsf{sb}}$)

from a-A a-all-acq a-owns
have a-notin′: a $\notin \mathcal{O} \cup$ A $-$ R $\cup$ all-acquired sb
by auto
from a-ro L-A a-A R-owns a-owns
have a $\in$ read-only ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (auto simp add: in-read-only-convs split: if-split-asm)

from Cons.hyps [OF no-wrt consis′ this a-notin′]
have a $\notin$ outstanding-refs is-Write$_{\mathsf{sb}}$ sb**.**
then
show ?thesis
by (clarsimp simp add: Ghost$_{\mathsf{sb}}$)
qed
qed

lemma read-only-reads-read-only:
$\bigwedge\mathcal{S}\ \mathcal{O}.$ $[\![$non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb;
sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb$]\!]$
$\Longrightarrow$
read-only-reads $\mathcal{O}$ sb $\subseteq$ $\mathcal{O}$ $\cup$ all-acquired sb $\cup$ read-only $\mathcal{S}$
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case (Write$_{sb}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)
case True

from Cons.prems **obtain** non-vol: non-volatile-owned-or-read-only True ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) ($\mathcal{O}$ $\cup$ A $-$ R) sb **and**
A-shared-onws: A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and**
consis$'$: sharing-consistent ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) ($\mathcal{O}$ $\cup$ A $-$ R) sb
by (clarsimp simp add: Write$_{sb}$ True)

from Cons.hyps [OF non-vol consis$'$]
have hyp: read-only-reads ($\mathcal{O}$ $\cup$ A $-$ R) sb
$\subseteq$ $\mathcal{O}$ $\cup$ A $-$ R $\cup$ all-acquired sb $\cup$ read-only ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L).

{
fix a$'$
assume a$'$-in: a$'$ $\in$ read-only-reads ($\mathcal{O}$ $\cup$ A $-$ R) sb
assume a$'$-unowned: a$'$ $\notin$ $\mathcal{O}$
assume a$'$-unacq: a$'$ $\notin$ all-acquired sb
assume a$'$-A: a$'$ $\notin$ A
have a$'$ $\in$ read-only $\mathcal{S}$
proof $-$
from a$'$-in hyp a$'$-unowned a$'$-unacq a$'$-A
have a$'$ $\in$ read-only ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)
by auto

with L-A R-owns a$'$-unowned
show ?thesis
by (auto simp add: in-read-only-convs split:if-split-asm)
qed
}

then

show ?thesis
apply (clarsimp simp add: Write$_{sb}$ True simp del: o-apply)

apply force
done
next
case False **with** Cons **show** ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$)
qed
next
case $\text{Read}_{\mathsf{sb}}$ **with** Cons **show** ?thesis
by auto
next
case $\text{Prog}_{\mathsf{sb}}$ **with** Cons **show** ?thesis
by auto
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.prems **obtain** non-vol: non-volatile-owned-or-read-only True ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb **and**
A-shared-onws: A $\subseteq$ dom $\mathcal{S} \cup \mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq \mathcal{O}$ **and**
consis$'$: sharing-consistent ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)

from Cons.hyps [OF non-vol consis$'$]
have hyp: read-only-reads ($\mathcal{O} \cup$ A $-$ R) sb
$\subseteq \mathcal{O} \cup$ A $-$ R $\cup$ all-acquired sb $\cup$ read-only ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L).

{
fix a$'$
assume a$'$-in: a$' \in$ read-only-reads ($\mathcal{O} \cup$ A $-$ R) sb
assume a$'$-unowned: a$' \notin \mathcal{O}$
assume a$'$-unacq: a$' \notin$ all-acquired sb
assume a$'$-A: a$' \notin$ A
have a$' \in$ read-only $\mathcal{S}$
proof $-$
from a$'$-in hyp a$'$-unowned a$'$-unacq a$'$-A
have a$' \in$ read-only ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by auto

with L-A R-owns a$'$-unowned
show ?thesis
by (auto simp add: in-read-only-convs split:if-split-asm)
qed
}

then

show ?thesis
apply (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$ simp del: o-apply)
apply force
done

qed
qed

lemma no-unacquired-write-to-read-only-reads:
$\bigwedge\mathcal{S}\ \mathcal{O}$. [[no-write-to-read-only-memory $\mathcal{S}$ sb;
non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb; sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb;
a $\in$ read-only-reads $\mathcal{O}$ sb; a $\notin$ ($\mathcal{O}$ $\cup$ all-acquired sb)]]
$\Longrightarrow$ a $\notin$ outstanding-refs is-Write$_{sb}$ sb
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case (Write$_{sb}$ volatile a′ sop v A L R W)
show ?thesis
proof (cases volatile)
case True

from Cons.prems **obtain** no-wrt: no-write-to-read-only-memory ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) sb
and
non-vol: non-volatile-owned-or-read-only True ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) ($\mathcal{O}$ $\cup$ A $-$ R) sb **and**
A-shared-onws: A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns:
R $\subseteq$ $\mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) ($\mathcal{O}$ $\cup$ A $-$ R) sb **and**
a-ro: a $\in$ read-only-reads ($\mathcal{O}$ $\cup$ A $-$ R) sb **and**
a-A: a $\notin$ A **and** a-all-acq: a $\notin$ all-acquired sb **and** a-owns: a $\notin$ $\mathcal{O}$ **and**
a′-notin: a′ $\notin$ read-only $\mathcal{S}$
by (simp add: Write$_{sb}$ True)

from read-only-reads-read-only [OF non-vol consis′] a-ro a-owns a-all-acq a-A
have a $\in$ read-only ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)
by auto
with a′-notin R-owns a-owns **have** neq-a-a′: a$\neq$a′
by (auto simp add: in-read-only-convs split: if-split-asm)

from a-A a-all-acq a-owns
have a-notin′: a $\notin$ $\mathcal{O}$ $\cup$ A $-$ R $\cup$ all-acquired sb
by auto

from Cons.hyps [OF no-wrt non-vol consis′ a-ro a-notin′]
have a $\notin$ outstanding-refs is-Write$_{sb}$ sb**.**
then
show ?thesis
using neq-a-a′
by (auto simp add: Write$_{sb}$ True)
next
case False **with** Cons
show ?thesis

by (auto simp add: $\text{Write}_{\mathsf{sb}}$ False)
  qed
 next
  case ($\text{Read}_{\mathsf{sb}}$ volatile a′ t v)
  show ?thesis
  proof (cases volatile)
    case True
    with Cons **show** ?thesis
by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
  next
    case False
    note non-volatile = this
    from Cons.prems **obtain** no-wrt′: no-write-to-read-only-memory $\mathcal{S}$ sb **and**
consis′:sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb **and**
a-in: a ∈ (if a′ ∉ $\mathcal{O}$ then insert a′ (read-only-reads $\mathcal{O}$ sb)
        else read-only-reads $\mathcal{O}$ sb) **and**
a′-owns-shared: a′ ∈ $\mathcal{O}$ ∨ a′ ∈ read-only $\mathcal{S}$ **and**
non-vol′: non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb **and**
    a-owns: a ∉ $\mathcal{O}$ ∪ all-acquired sb
by (clarsimp simp add: $\text{Read}_{\mathsf{sb}}$ False)

    show ?thesis
    proof (cases a′ ∈ $\mathcal{O}$)
case True
with a-in **have** a ∈ read-only-reads $\mathcal{O}$ sb
 by auto
from Cons.hyps [OF no-wrt′ non-vol′ consis′ this a-owns]
show ?thesis
 by (clarsimp simp add: $\text{Read}_{\mathsf{sb}}$)
    next
case False
note a′-unowned = this
with a-in **have** a-in′: a ∈ insert a′ (read-only-reads $\mathcal{O}$ sb) **by** auto
from a′-owns-shared False **have** a′-read-only: a′ ∈ read-only $\mathcal{S}$ **by** auto
show ?thesis
proof (cases a=a′)
 case False
 with a-in′ **have** a ∈ (read-only-reads $\mathcal{O}$ sb) **by** auto
 from Cons.hyps [OF no-wrt′ non-vol′ consis′ this a-owns]
 show ?thesis
  by (simp add: $\text{Read}_{\mathsf{sb}}$)
next
 case True
 from no-unacquired-write-to-read-only [OF no-wrt′ consis′ a′-read-only] a-owns True

 have a′ ∉ outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ sb
  by auto
 then show ?thesis
  by (simp add: $\text{Read}_{\mathsf{sb}}$ True)
qed

qed
qed
next
case $\text{Prog}_{\mathsf{sb}}$ **with** Cons
show ?thesis
by (auto)
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.prems **obtain** no-wrt: no-write-to-read-only-memory ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) sb **and**
non-vol: non-volatile-owned-or-read-only True ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb **and**
A-shared-onws: A $\subseteq$ dom $\mathcal{S} \cup \mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq \mathcal{O}$ **and**
consis$'$: sharing-consistent ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb **and**
a-ro: a $\in$ read-only-reads ($\mathcal{O} \cup$ A $-$ R) sb **and**
a-A: a $\notin$ A **and** a-all-acq: a $\notin$ all-acquired sb **and** a-owns: a $\notin \mathcal{O}$
by (simp add: $\text{Ghost}_{\mathsf{sb}}$)

from read-only-reads-read-only [OF non-vol consis$'$] a-ro a-owns a-all-acq a-A
have a $\in$ read-only ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by auto

from a-A a-all-acq a-owns
have a-notin$'$: a $\notin \mathcal{O} \cup$ A $-$ R $\cup$ all-acquired sb
by auto

from Cons.hyps [OF no-wrt non-vol consis$'$ a-ro a-notin$'$]
have a $\notin$ outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ sb**.**
then
show ?thesis
by (auto simp add: $\text{Ghost}_{\mathsf{sb}}$)
qed
qed

lemma no-unacquired-write-to-read-only$''$:
assumes no-wrt: no-write-to-read-only-memory $\mathcal{S}$ sb
assumes consis: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb
shows read-only $\mathcal{S} \cap$ outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ sb $\subseteq \mathcal{O} \cup$ all-acquired sb
using no-unacquired-write-to-read-only [OF no-wrt consis]
by auto

lemma no-unacquired-volatile-write-to-read-only:
assumes no-wrt: no-write-to-read-only-memory $\mathcal{S}$ sb
assumes consis: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb
shows read-only $\mathcal{S} \cap$ outstanding-refs is-volatile-$\text{Write}_{\mathsf{sb}}$ sb $\subseteq \mathcal{O} \cup$ all-acquired sb
proof $-$
have outstanding-refs is-volatile-$\text{Write}_{\mathsf{sb}}$ sb $\subseteq$ outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ sb
apply (rule outstanding-refs-mono-pred)

apply (auto simp add: is-volatile-Write$_{sb}$-def split: memref.splits)
done
with no-unacquired-write-to-read-only$''$ [OF no-wrt consis]
show ?thesis **by** blast
qed

lemma no-unacquired-non-volatile-write-to-read-only-reads:
assumes no-wrt: no-write-to-read-only-memory $\mathcal{S}$ sb
assumes consis: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb
shows read-only $\mathcal{S}$ $\cap$ outstanding-refs is-non-volatile-Write$_{sb}$ sb $\subseteq$ $\mathcal{O}$ $\cup$ all-acquired sb
proof –
from outstanding-refs-subsets
have outstanding-refs is-non-volatile-Write$_{sb}$ sb $\subseteq$ outstanding-refs is-Write$_{sb}$ sb **by** – assumption
with no-unacquired-write-to-read-only$''$ [OF no-wrt consis]
show ?thesis **by** blast
qed

lemma no-unacquired-write-to-read-only-reads$'$:
assumes no-wrt: no-write-to-read-only-memory $\mathcal{S}$ sb
assumes non-vol: non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb
assumes consis: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb
shows read-only-reads $\mathcal{O}$ sb $\cap$ outstanding-refs is-Write$_{sb}$ sb $\subseteq$ $\mathcal{O}$ $\cup$ all-acquired sb
using no-unacquired-write-to-read-only-reads [OF no-wrt non-vol consis]
by auto

lemma no-unacquired-volatile-write-to-read-only-reads:
assumes no-wrt: no-write-to-read-only-memory $\mathcal{S}$ sb
assumes non-vol: non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb
assumes consis: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb
shows read-only-reads $\mathcal{O}$ sb $\cap$ outstanding-refs is-volatile-Write$_{sb}$ sb $\subseteq$ $\mathcal{O}$ $\cup$ all-acquired sb
proof –
have outstanding-refs is-volatile-Write$_{sb}$ sb $\subseteq$ outstanding-refs is-Write$_{sb}$ sb
apply (rule outstanding-refs-mono-pred)
apply (auto simp add: is-volatile-Write$_{sb}$-def split: memref.splits)
done
with no-unacquired-write-to-read-only-reads [OF no-wrt non-vol consis]
show ?thesis **by** blast
qed

lemma no-unacquired-non-volatile-write-to-read-only:
assumes no-wrt: no-write-to-read-only-memory $\mathcal{S}$ sb
assumes non-vol: non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb
assumes consis: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb
shows read-only-reads $\mathcal{O}$ sb $\cap$ outstanding-refs is-non-volatile-Write$_{sb}$ sb $\subseteq$ $\mathcal{O}$ $\cup$ all-acquired sb
proof –
from outstanding-refs-subsets

have outstanding-refs is-non-volatile-Write$_{sb}$ sb $\subseteq$ outstanding-refs is-Write$_{sb}$ sb **by** – assumption
with no-unacquired-write-to-read-only-reads [OF no-wrt non-vol consis]
show ?thesis **by** blast
qed

lemma set-dropWhileD: x $\in$ set (dropWhile P xs) $\Longrightarrow$ x $\in$ set xs
by (induct xs) (auto split: if-split-asm)

lemma outstanding-refs-takeWhileD:
x $\in$ outstanding-refs P (takeWhile P$'$ sb) $\Longrightarrow$ x $\in$ outstanding-refs P sb
using outstanding-refs-takeWhile
by blast

lemma outstanding-refs-dropWhileD:
x $\in$ outstanding-refs P (dropWhile P$'$ sb) $\Longrightarrow$ x $\in$ outstanding-refs P sb
by (auto dest: set-dropWhileD simp add: outstanding-refs-conv)

lemma dropWhile-ConsD: dropWhile P xs = y#ys $\Longrightarrow$ $\neg$ P y
by (simp add: dropWhile-eq-Cons-conv)

lemma non-volatile-owned-or-read-only-drop:
non-volatile-owned-or-read-only False $\mathcal{S}$ $\mathcal{O}$ sb
$\Longrightarrow$ non-volatile-owned-or-read-only True
(share (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\mathcal{S}$)
(acquired True (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\mathcal{O}$)
(dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb)
using non-volatile-owned-or-read-only-append [of False $\mathcal{S}$ $\mathcal{O}$ (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb)
(dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb)]
apply (cases outstanding-refs is-volatile-Write$_{sb}$ sb = {})
apply (clarsimp simp add: outstanding-vol-write-take-drop-appends
takeWhile-not-vol-write-outstanding-refs dropWhile-not-vol-write-empty)
apply(clarsimp simp add: outstanding-vol-write-take-drop-appends
takeWhile-not-vol-write-outstanding-refs dropWhile-not-vol-write-empty)
apply (case-tac (dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb))
apply (fastforce simp add: outstanding-refs-conv)
apply (frule dropWhile-ConsD)
apply (clarsimp split: memref.splits)
done

lemma read-only-share: $\bigwedge\mathcal{S}$ $\mathcal{O}$.
sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb $\Longrightarrow$
read-only (share sb $\mathcal{S}$) $\subseteq$ read-only $\mathcal{S}$ $\cup$ $\mathcal{O}$ $\cup$ all-acquired sb

proof (induct sb)
case Nil **thus** ?case **by** auto
next
case (Cons x sb)
show ?case
proof (cases x)
case ($Write_{sb}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)
case True
from Cons.prems **obtain**
A-shared-owns: A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) ($\mathcal{O}$ $\cup$ A $-$ R) sb
by (clarsimp simp add: $Write_{sb}$ True)
from Cons.hyps [OF consis′]
have read-only (share sb ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L))
$\subseteq$ read-only ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) $\cup$ ($\mathcal{O}$ $\cup$ A $-$ R) $\cup$ all-acquired sb
by auto
also from A-shared-owns L-A R-owns A-R
have read-only ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) $\cup$ ($\mathcal{O}$ $\cup$ A $-$ R) $\cup$ all-acquired sb $\subseteq$
read-only $\mathcal{S}$ $\cup$ $\mathcal{O}$ $\cup$ (A $\cup$ all-acquired sb)
by (auto simp add: read-only-def augment-shared-def restrict-shared-def split: option.splits)
finally
show ?thesis
by (simp add: $Write_{sb}$ True)
next
case False **with** Cons **show** ?thesis
by (auto simp add: $Write_{sb}$)
qed
next
case $Read_{sb}$ **with** Cons **show** ?thesis
by auto
next
case $Prog_{sb}$ **with** Cons **show** ?thesis
by auto
next
case ($Ghost_{sb}$ A L R W)
from Cons.prems **obtain**
A-shared-owns: A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) ($\mathcal{O}$ $\cup$ A $-$ R) sb
by (clarsimp simp add: $Ghost_{sb}$)
from Cons.hyps [OF consis′]
have read-only (share sb ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L))
$\subseteq$ read-only ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) $\cup$ ($\mathcal{O}$ $\cup$ A $-$ R) $\cup$ all-acquired sb
by auto
also from A-shared-owns L-A R-owns A-R
have read-only ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) $\cup$ ($\mathcal{O}$ $\cup$ A $-$ R) $\cup$ all-acquired sb $\subseteq$

read-only $\mathcal{S} \cup \mathcal{O} \cup (A \cup$ all-acquired sb)
by (auto simp add: read-only-def augment-shared-def restrict-shared-def split: option.splits)
finally
show ?thesis
by (simp add: $\mathrm{Ghost}_{\mathsf{sb}}$)
qed
qed

lemma (**in** valid-ownership-and-sharing) outstanding-non-write-non-vol-reads-drop-disj:
assumes i-bound: $i <$ length ts
assumes j-bound: $j <$ length ts
assumes neq-i-j: $i \neq j$
assumes ith: ts!i = $(p_i,is_i,j_i,sb_i,\mathcal{D}_i,\mathcal{O}_i,\mathcal{R}_i)$
assumes jth: ts!j = $(p_j,is_j,j_j,sb_j,\mathcal{D}_j,\mathcal{O}_j,\mathcal{R}_j)$
shows outstanding-refs is-$\mathrm{Write}_{\mathsf{sb}}$ (dropWhile (Not $\circ$ is-volatile-$\mathrm{Write}_{\mathsf{sb}}$) sb_i) $\cap$
outstanding-refs is-non-volatile-$\mathrm{Read}_{\mathsf{sb}}$ (dropWhile (Not $\circ$ is-volatile-$\mathrm{Write}_{\mathsf{sb}}$) sb_j)
= {}
proof −

let ?take-j = (takeWhile (Not $\circ$ is-volatile-$\mathrm{Write}_{\mathsf{sb}}$) sb_j)
let ?drop-j = (dropWhile (Not $\circ$ is-volatile-$\mathrm{Write}_{\mathsf{sb}}$) sb_j)

let ?take-i = (takeWhile (Not $\circ$ is-volatile-$\mathrm{Write}_{\mathsf{sb}}$) sb_i)
let ?drop-i = (dropWhile (Not $\circ$ is-volatile-$\mathrm{Write}_{\mathsf{sb}}$) sb_i)

note nvo-i = outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ith]
note nvo-j = outstanding-non-volatile-refs-owned-or-read-only [OF j-bound jth]
note nro-i = no-outstanding-write-to-read-only-memory [OF i-bound ith]
with no-write-to-read-only-memory-append [of $\mathcal{S}$?take-i ?drop-i]
have nro-drop-i: no-write-to-read-only-memory (share ?take-i $\mathcal{S}$) ?drop-i
by simp
note nro-j = no-outstanding-write-to-read-only-memory [OF j-bound jth]
with no-write-to-read-only-memory-append [of $\mathcal{S}$?take-j ?drop-j]
have nro-drop-j: no-write-to-read-only-memory (share ?take-j $\mathcal{S}$) ?drop-j
by simp
from outstanding-volatile-writes-unowned-by-others [OF i-bound j-bound neq-i-j ith jth]
have dist: $(\mathcal{O}_j \cup$ all-acquired $sb_j) \cap$ outstanding-refs is-volatile-$\mathrm{Write}_{\mathsf{sb}}$ sb_i = {}.
note own-dist = ownership-distinct [OF i-bound j-bound neq-i-j ith jth]

from sharing-consis [OF j-bound jth]
have consis-j: sharing-consistent $\mathcal{S}$ $\mathcal{O}_j$ sb_j.
with sharing-consistent-append [of $\mathcal{S}$ $\mathcal{O}_j$?take-j ?drop-j]
obtain

consis-take-j: sharing-consistent $\mathcal{S}$ $\mathcal{O}_j$?take-j **and**
consis-drop-j: sharing-consistent (share ?take-j $\mathcal{S}$) (acquired True ?take-j $\mathcal{O}_j$) ?drop-j
by simp

from sharing-consis [OF i-bound ith]
have consis-i: sharing-consistent $\mathcal{S}$ $\mathcal{O}_i$ sb_i**.**
with sharing-consistent-append [of $\mathcal{S}$ $\mathcal{O}_i$?take-i ?drop-i]
have consis-drop-i: sharing-consistent (share ?take-i $\mathcal{S}$) (acquired True ?take-i $\mathcal{O}_i$) ?drop-i
by simp

{
fix x
assume x-in-drop-i: x $\in$ outstanding-refs is-Write$_{sb}$?drop-i
assume x-in-drop-j: x $\in$ outstanding-refs is-non-volatile-Read$_{sb}$?drop-j
have False
proof –
from x-in-drop-i **have** x-in-i: x $\in$ outstanding-refs is-Write$_{sb}$ sb_i
using outstanding-refs-append [of is-Write$_{sb}$?take-i ?drop-i] **by** auto

from x-in-drop-j **have** x-in-j: x $\in$ outstanding-refs is-non-volatile-Read$_{sb}$ sb_j
using outstanding-refs-append [of is-non-volatile-Read$_{sb}$?take-j ?drop-j]
by auto
from non-volatile-owned-or-read-only-drop [OF nvo-j]
have nvo-drop-j: non-volatile-owned-or-read-only True (share ?take-j $\mathcal{S}$) (acquired True ?take-j $\mathcal{O}_j$) ?drop-j**.**

from non-volatile-reads-acquired-or-read-only-reads [OF nvo-drop-j] x-in-drop-j
acquired-takeWhile-non-volatile-Write$_{sb}$ [of sb_j $\mathcal{O}_j$]
have x-j: x $\in$ $\mathcal{O}_j$ $\cup$ all-acquired sb_j $\cup$ read-only-reads (acquired True ?take-j $\mathcal{O}_j$) ?drop-j
using all-acquired-append [of ?take-j ?drop-j]
by (auto)

{
assume x-in-vol-drop-i: x $\in$ outstanding-refs is-volatile-Write$_{sb}$?drop-i
hence x-in-vol-i: x $\in$ outstanding-refs is-volatile-Write$_{sb}$ sb_i
using outstanding-refs-append [of is-volatile-Write$_{sb}$?take-i ?drop-i]
by auto

from outstanding-volatile-writes-unowned-by-others [OF i-bound j-bound neq-i-j ith jth]
have ($\mathcal{O}_j$ $\cup$ all-acquired sb_j) $\cap$ outstanding-refs is-volatile-Write$_{sb}$ sb_i = {}**.**
with x-in-vol-i x-j **obtain**
x-unacq-j: x $\notin$ $\mathcal{O}_j$ $\cup$ all-acquired sb_j **and**
x-ror-j: x $\in$ read-only-reads (acquired True ?take-j $\mathcal{O}_j$) ?drop-j
by auto
from read-only-reads-unowned [OF j-bound i-bound neq-i-j [symmetric] jth ith] x-ror-j
have x $\notin$ $\mathcal{O}_i$ $\cup$ all-acquired sb_i
by auto

moreover

from read-only-reads-read-only [OF nvo-drop-j consis-drop-j] x-ror-j x-unacq-j
all-acquired-append [of ?take-j ?drop-j] acquired-takeWhile-non-volatile-Write$_{sb}$ [of sb$_j$
$\mathcal{O}_j$]
have x ∈ read-only (share ?take-j $\mathcal{S}$)
by (auto)

from read-only-share [OF consis-take-j] this x-unacq-j all-acquired-append [of ?take-j
?drop-j]
have x ∈ read-only $\mathcal{S}$
by auto

with no-unacquired-write-to-read-only″ [OF nro-i consis-i] x-in-i
have x ∈ $\mathcal{O}_i$ ∪ all-acquired sb$_i$
by auto

ultimately have False **by** auto
}
moreover
{
assume x-in-non-vol-drop-i: x ∈ outstanding-refs is-non-volatile-Write$_{sb}$?drop-i
hence x ∈ outstanding-refs is-non-volatile-Write$_{sb}$ sb$_i$
using outstanding-refs-append [of is-non-volatile-Write$_{sb}$?take-i ?drop-i]
by auto
with non-volatile-owned-or-read-only-outstanding-non-volatile-writes [OF nvo-i]
have x ∈ $\mathcal{O}_i$ ∪ all-acquired sb$_i$ **by** auto

moreover

with x-j own-dist **obtain**
x-unacq-j: x ∉ $\mathcal{O}_j$ ∪ all-acquired sb$_j$ **and**
x-ror-j: x ∈ read-only-reads (acquired True ?take-j $\mathcal{O}_j$) ?drop-j
by auto
from read-only-reads-unowned [OF j-bound i-bound neq-i-j [symmetric] jth ith] x-ror-j
have x ∉ $\mathcal{O}_i$ ∪ all-acquired sb$_i$
by auto

ultimately have False
by auto
}
ultimately

show ?thesis
using x-in-drop-i x-in-drop-j
by (auto simp add: misc-outstanding-refs-convs)
qed
}
thus ?thesis

by auto
qed

lemma (**in** valid-ownership-and-sharing) outstanding-non-volatile-write-disj:
assumes i-bound: $i < \text{length ts}$
assumes j-bound: $j < \text{length ts}$
assumes neq-i-j: $i \neq j$
assumes ith: ts!i = $(p_i,is_i,j_i,sb_i,\mathcal{D}_i,\mathcal{O}_i,\mathcal{R}_i)$
assumes jth: ts!j = $(p_j,is_j,j_j,sb_j,\mathcal{D}_j,\mathcal{O}_j,\mathcal{R}_j)$
shows outstanding-refs (is-non-volatile-Write$_{sb}$) (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_i) $\cap$
(outstanding-refs is-volatile-Write$_{sb}$ sb_j $\cup$
outstanding-refs is-non-volatile-Write$_{sb}$ sb_j $\cup$
outstanding-refs is-non-volatile-Read$_{sb}$ (dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j) $\cup$
(outstanding-refs is-non-volatile-Read$_{sb}$ (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j) $-$
read-only-reads $\mathcal{O}_j$ (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j)) $\cup$
($\mathcal{O}_j$ $\cup$ all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j))
) = {} (**is** ?non-vol-writes-i $\cap$?not-volatile-j = {})
proof $-$
note nro-i = no-outstanding-write-to-read-only-memory [OF i-bound ith]
note nro-j = no-outstanding-write-to-read-only-memory [OF j-bound jth]
note nvo-j = outstanding-non-volatile-refs-owned-or-read-only [OF j-bound jth]
note nvo-i = outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ith]

from outstanding-volatile-writes-unowned-by-others [OF i-bound j-bound neq-i-j ith jth]
have dist: ($\mathcal{O}_j$ $\cup$ all-acquired sb_j) $\cap$ outstanding-refs is-volatile-Write$_{sb}$ sb_i = {}.

from outstanding-volatile-writes-unowned-by-others [OF j-bound i-bound neq-i-j [symmetric] jth ith]
have dist-j: ($\mathcal{O}_i$ $\cup$ all-acquired sb_i) $\cap$ outstanding-refs is-volatile-Write$_{sb}$ sb_j = {}.

note own-dist = ownership-distinct [OF i-bound j-bound neq-i-j ith jth]

from sharing-consis [OF j-bound jth]
have consis-j: sharing-consistent $\mathcal{S}$ $\mathcal{O}_j$ sb_j.

from sharing-consis [OF i-bound ith]
have consis-i: sharing-consistent $\mathcal{S}$ $\mathcal{O}_i$ sb_i.

let ?take-j = (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j)
let ?drop-j = (dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j)

{
fix x
assume x-in-take-i: $x \in$?non-vol-writes-i
assume x-in-j: $x \in$?not-volatile-j
from x-in-take-i **have** x-in-i: $x \in$ outstanding-refs (is-non-volatile-Write$_{sb}$) sb_i

by (auto dest: outstanding-refs-takeWhileD)
from non-volatile-owned-or-read-only-outstanding-non-volatile-writes [OF nvo-i] x-in-i
have x-in-owns-acq-i: x $\in \mathcal{O}_{\mathsf{i}} \cup$ all-acquired sb_{i}
by auto
have False
proof −
{
assume x-in-j: x $\in$ outstanding-refs is-volatile-$Write_{\mathsf{sb}}$ sb_{j}
with dist-j **have** x-notin: x $\notin$ ($\mathcal{O}_{\mathsf{i}} \cup$ all-acquired sb_{i})
by auto
with x-in-owns-acq-i **have** False
by auto
}
moreover
{
assume x-in-j: x $\in$ outstanding-refs is-non-volatile-$Write_{\mathsf{sb}}$ sb_{j}
from non-volatile-owned-or-read-only-outstanding-non-volatile-writes [OF nvo-j] x-in-j
have x $\in \mathcal{O}_{\mathsf{j}} \cup$ all-acquired sb_{j}
by auto
with x-in-owns-acq-i own-dist
have False
by auto
}
moreover
{
assume x-in-j: x $\in$ outstanding-refs is-non-volatile-$Read_{\mathsf{sb}}$?drop-j

from non-volatile-owned-or-read-only-drop [OF nvo-j]
have nvo$'$: non-volatile-owned-or-read-only True (share ?take-j $\mathcal{S}$) (acquired True ?take-j $\mathcal{O}_{\mathsf{j}}$) ?drop-j**.**

from non-volatile-owned-or-read-only-outstanding-refs [OF nvo$'$] x-in-j
have x $\in$ acquired True ?take-j $\mathcal{O}_{\mathsf{j}} \cup$ all-acquired ?drop-j $\cup$
read-only-reads (acquired True ?take-j $\mathcal{O}_{\mathsf{j}}$) ?drop-j
by (auto simp add: misc-outstanding-refs-convs)

moreover
from acquired-append [of True ?take-j ?drop-j $\mathcal{O}_{\mathsf{j}}$] acquired-all-acquired [of True ?take-j $\mathcal{O}_{\mathsf{j}}$]
all-acquired-append [of ?take-j ?drop-j]
have acquired True ?take-j $\mathcal{O}_{\mathsf{j}} \cup$ all-acquired ?drop-j $\subseteq \mathcal{O}_{\mathsf{j}} \cup$ all-acquired sb_{j}
by auto
ultimately
have x $\in$ read-only-reads (acquired True ?take-j $\mathcal{O}_{\mathsf{j}}$) ?drop-j
using x-in-owns-acq-i own-dist
by auto

with read-only-reads-unowned [OF j-bound i-bound neq-i-j [symmetric] jth ith] x-in-owns-acq-i
have False

by auto
}
moreover
{
assume x-in-j: x ∈ outstanding-refs is-non-volatile-Read$_{sb}$?take-j
assume x-notin: x ∉ read-only-reads $\mathcal{O}_j$?take-j
from non-volatile-owned-or-read-only-append [**where** xs=?take-j **and** ys=?drop-j] nvo-j
have non-volatile-owned-or-read-only False $\mathcal{S}$ $\mathcal{O}_j$?take-j
by auto

from non-volatile-owned-or-read-only-outstanding-refs [OF this] x-in-j x-notin
have x ∈ $\mathcal{O}_j$ ∪ all-acquired ?take-j
by (auto simp add: misc-outstanding-refs-convs)
with all-acquired-append [of ?take-j ?drop-j] x-in-owns-acq-i own-dist
have False
by auto
}
moreover
{
assume x-in-j: x ∈ $\mathcal{O}_j$ ∪ all-acquired ?take-j
moreover
from all-acquired-append [of ?take-j ?drop-j]
have all-acquired ?take-j ⊆ all-acquired sb$_j$
by auto
ultimately have False
using x-in-owns-acq-i own-dist
by auto
}
ultimately show ?thesis
using x-in-take-i x-in-j
by (auto simp add: misc-outstanding-refs-convs)
qed
}
then show ?thesis
by auto
qed

lemma (**in** valid-ownership-and-sharing) outstanding-non-volatile-write-not-volatile-read-disj:
assumes i-bound: i < length ts
assumes j-bound: j < length ts
assumes neq-i-j: i ≠ j
assumes ith: ts!i = (p$_i$,is$_i$,j$_i$,sb$_i$,$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$)
assumes jth: ts!j = (p$_j$,is$_j$,j$_j$,sb$_j$,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
shows outstanding-refs (is-non-volatile-Write$_{sb}$) (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_i$) ∩
outstanding-refs (Not ∘ is-volatile-Read$_{sb}$) (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$) = {}
(**is** ?non-vol-writes-i ∩ ?not-volatile-j = {})
proof −

have outstanding-refs (Not ∘ is-volatile-Read$_{sb}$) (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$) ⊆
outstanding-refs is-volatile-Write$_{sb}$ sb$_j$ ∪
outstanding-refs is-non-volatile-Write$_{sb}$ sb$_j$ ∪
outstanding-refs is-non-volatile-Read$_{sb}$ (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$)
by (auto simp add: misc-outstanding-refs-convs dest: outstanding-refs-dropWhileD)
with outstanding-non-volatile-write-disj [OF i-bound j-bound neq-i-j ith jth]
show ?thesis
by blast
qed

lemma (**in** valid-ownership-and-sharing) outstanding-refs-is-Write$_{sb}$-takeWhile-disj:
∀ i < length ts. (∀ j < length ts. i ≠ j ⟶
(let (-,-,-,sb$_i$,-,-,-) = ts!i;
(-,-,-,sb$_j$,-,-,-) = ts!j
in outstanding-refs is-Write$_{sb}$ sb$_i$ ∩
outstanding-refs is-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$) = {}))
proof −
{
fix i j p$_i$ is$_i$ $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j$_i$ sb$_i$ p$_j$ is$_j$ $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j$_j$ sb$_j$
assume i-bound: i < length ts
assume j-bound: j < length ts
assume neq-i-j: i ≠ j
assume ith: ts!i = (p$_i$,is$_i$,j$_i$,sb$_i$,$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$)
assume jth: ts!j = (p$_j$,is$_j$,j$_j$,sb$_j$,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
from outstanding-non-volatile-write-disj [OF j-bound i-bound neq-i-j[symmetric] jth ith]
have outstanding-refs is-Write$_{sb}$ sb$_i$ ∩
outstanding-refs is-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$) = {}
apply (clarsimp simp add: outstanding-refs-is-non-volatile-Write$_{sb}$-takeWhile-conv)
apply (auto simp add: misc-outstanding-refs-convs)
done
}
thus ?thesis
by (fastforce simp add: Let-def)
qed

fun read-tmps:: ′p store-buffer ⇒ tmp set
where
read-tmps [] = {}
| read-tmps (r#rs) =
(case r of
Read$_{sb}$ volatile a t v ⇒ insert t (read-tmps rs)
| - ⇒ read-tmps rs)

lemma in-read-tmps-conv:
  (t ∈ read-tmps xs) = (∃ volatile a v. $\text{Read}_{\mathsf{sb}}$ volatile a t v ∈ set xs)
  by (induct xs) (auto split: memref.splits)

lemma read-tmps-mono: ⋀ys. set xs ⊆ set ys ⟹ read-tmps xs ⊆ read-tmps ys
  by (fastforce simp add: in-read-tmps-conv)

fun distinct-read-tmps:: 'p store-buffer ⇒ bool
where
  distinct-read-tmps [] = True
| distinct-read-tmps (r#rs) =
    (case r of
      $\text{Read}_{\mathsf{sb}}$ volatile a t v ⇒ t ∉ (read-tmps rs) ∧ distinct-read-tmps rs
     | - ⇒ distinct-read-tmps rs)

lemma distinct-read-tmps-conv:
 distinct-read-tmps xs = (∀ i < length xs. ∀ j < length xs. i ≠ j ⟶
    (case xs!i of
      $\text{Read}_{\mathsf{sb}}$ - - t_i - ⇒ case xs!j of $\text{Read}_{\mathsf{sb}}$ - - t_j - ⇒ $t_i \neq t_j$ | - ⇒ True
     | - ⇒ True))
— Nice lemma, ugly proof.
proof (induct xs)
  case Nil **thus** ?case **by** simp
next
  case (Cons x xs)
  show ?case
  proof (cases x)
    case ($\text{Write}_{\mathsf{sb}}$ volatile a sop v)
    with Cons.hyps **show** ?thesis
      apply –
      apply (rule iffI [rule-format])
      apply clarsimp
          subgoal for i j
          apply (case-tac i)
          apply fastforce
          apply (case-tac j)
          apply (fastforce split: memref.splits)
          apply (clarsimp cong: memref.case-cong)
          done
      apply clarsimp
      subgoal for i j
      apply (erule-tac x=Suc i **in** allE)
      apply clarsimp
      apply (erule-tac x=Suc j **in** allE)
      apply (clarsimp cong: memref.case-cong)
      done

```
    done
next
  case (Read_sb volatile a t v)
  with Cons.hyps show ?thesis
    apply –
    apply (rule iffI [rule-format])
    apply  clarsimp
           subgoal for i j
           apply (case-tac i)
           apply  clarsimp
           apply  (case-tac j)
           apply   clarsimp
        apply  (fastforce split: memref.splits simp add: in-read-tmps-conv dest: nth-mem)
           apply (clarsimp)
           apply (case-tac j)
        apply  (fastforce split: memref.splits simp add: in-read-tmps-conv dest: nth-mem)
           apply (clarsimp cong: memref.case-cong)
           done
    apply clarsimp
    apply (rule conjI)
    apply (clarsimp simp add: in-read-tmps-conv)
    apply  (erule-tac x=0 in allE)
    apply  (clarsimp simp add: in-set-conv-nth)
           subgoal for volatile' a' v' i
           apply  (erule-tac x=Suc i in allE)
           apply  clarsimp
           done
    apply clarsimp
    subgoal for i j
    apply (erule-tac x=Suc i in allE)
    apply clarsimp
    apply (erule-tac x=Suc j in allE)
    apply (clarsimp cong: memref.case-cong)
    done
    done
next
  case Prog_sb
  with Cons.hyps show ?thesis
    apply –
    apply (rule iffI [rule-format])
    apply  clarsimp
           subgoal for i j
           apply (case-tac i)
           apply  fastforce
           apply (case-tac j)
           apply  (fastforce split: memref.splits)
           apply (clarsimp cong: memref.case-cong)
           done
    apply clarsimp
    subgoal for i j
```

```
      apply (erule-tac x=Suc i in allE)
      apply  clarsimp
      apply (erule-tac x=Suc j in allE)
      apply  (clarsimp cong: memref.case-cong)
      done
      done
  next
    case Ghost_sb
    with Cons.hyps show ?thesis
      apply −
      apply (rule iffI [rule-format])
      apply  clarsimp
             subgoal for i j
             apply (case-tac i)
             apply  fastforce
             apply (case-tac j)
             apply  (fastforce split: memref.splits)
             apply (clarsimp cong: memref.case-cong)
             done
      apply clarsimp
      subgoal for i j
      apply (erule-tac x=Suc i in allE)
      apply  clarsimp
      apply (erule-tac x=Suc j in allE)
      apply (clarsimp cong: memref.case-cong)
      done
      done
  qed
qed

fun load-tmps:: instrs ⇒ tmp set
where
  load-tmps [] = {}
| load-tmps (i#is) =
     (case i of
        Read volatile a t ⇒ insert t (load-tmps is)
      | RMW - t - - - - - - -  ⇒ insert t (load-tmps is)
      | - ⇒ load-tmps is)

lemma in-load-tmps-conv:
  (t ∈ load-tmps xs) = ((∃volatile a. Read volatile a t ∈ set xs) ∨
                        (∃a sop cond ret A L R W. RMW a t sop cond ret A L R W ∈ set xs))
  by (induct xs) (auto split: instr.splits)

lemma load-tmps-mono: ⋀ys. set xs ⊆ set ys ⟹ load-tmps xs ⊆ load-tmps ys
  by (fastforce simp add: in-load-tmps-conv)

fun distinct-load-tmps:: instrs ⇒ bool
where
  distinct-load-tmps [] = True
```

| distinct-load-tmps (r#rs) =
  (case r of
    Read volatile a t ⇒ t ∉ (load-tmps rs) ∧ distinct-load-tmps rs
   | RMW a t sop cond ret A L R W ⇒ t ∉ (load-tmps rs) ∧ distinct-load-tmps rs
   | - ⇒ distinct-load-tmps rs)

locale load-tmps-distinct =
fixes ts::('p,'p store-buffer,bool,owns,rels) thread-config list
assumes load-tmps-distinct:
 $\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
 ⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟧
 ⟹
 distinct-load-tmps is

locale read-tmps-distinct =
fixes ts::('p,'p store-buffer,bool,owns,rels) thread-config list
assumes read-tmps-distinct:
 $\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
 ⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟧
 ⟹
 distinct-read-tmps sb

locale load-tmps-read-tmps-distinct =
fixes ts::('p,'p store-buffer,bool,owns,rels) thread-config list
assumes load-tmps-read-tmps-distinct:
 $\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
 ⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟧
 ⟹
 load-tmps is ∩ read-tmps sb = {}

locale tmps-distinct =
 load-tmps-distinct +
 read-tmps-distinct +
 load-tmps-read-tmps-distinct

lemma rev-read-tmps: read-tmps (rev xs) = read-tmps xs
 by (auto simp add: in-read-tmps-conv)

lemma rev-load-tmps: load-tmps (rev xs) = load-tmps xs
 by (auto simp add: in-load-tmps-conv)

lemma distinct-read-tmps-append: $\bigwedge$ys. distinct-read-tmps (xs @ ys) =
    (distinct-read-tmps xs ∧ distinct-read-tmps ys ∧
    read-tmps xs ∩ read-tmps ys = {})
by (induct xs) (auto split: memref.splits simp add: in-read-tmps-conv)

lemma distinct-load-tmps-append: $\bigwedge$ys. distinct-load-tmps (xs @ ys) =
    (distinct-load-tmps xs ∧ distinct-load-tmps ys ∧
    load-tmps xs ∩ load-tmps ys = {})

```
apply (induct xs)
apply (auto split: instr.splits simp add: in-load-tmps-conv)
done

lemma read-tmps-append: read-tmps (xs@ys) = (read-tmps xs ∪ read-tmps ys)
  by (fastforce simp add: in-read-tmps-conv)

lemma load-tmps-append: load-tmps (xs@ys) = (load-tmps xs ∪ load-tmps ys)
  by (fastforce simp add: in-load-tmps-conv)

fun write-sops:: 'p store-buffer ⇒ sop set
where
  write-sops [] = {}
| write-sops (r#rs) =
     (case r of
       Write_sb volatile a sop v - - - -⇒ insert sop (write-sops rs)
      | - ⇒ write-sops rs)

lemma in-write-sops-conv:
  (sop ∈ write-sops xs) = (∃volatile a v A L R W. Write_sb volatile a sop v A L R W ∈ set
xs)
  apply (induct xs)
  apply  simp
  apply (auto split: memref.splits)
  apply  force
  apply force
  done

lemma write-sops-mono: ⋀ys. set xs ⊆ set ys ⟹ write-sops xs ⊆ write-sops ys
  by (fastforce simp add: in-write-sops-conv)

lemma write-sops-append: write-sops (xs@ys) = write-sops xs ∪ write-sops ys
  by (force simp add: in-write-sops-conv)


fun store-sops:: instrs ⇒ sop set
where
  store-sops [] = {}
| store-sops (i#is) =
     (case i of
        Write volatile a sop - - - - ⇒ insert sop (store-sops is)
      | RMW a t sop cond ret A L R W ⇒ insert sop (store-sops is)
      | - ⇒ store-sops is)

lemma in-store-sops-conv:
  (sop ∈ store-sops xs) = ((∃volatile a A L R W. Write volatile a sop A L R W ∈ set xs)
∨
                         (∃a t cond ret A L R W. RMW a t sop cond ret A L R W ∈ set xs))
  by (induct xs) (auto split: instr.splits)
```

lemma store-sops-mono: $\bigwedge$ys. set xs $\subseteq$ set ys $\Longrightarrow$ store-sops xs $\subseteq$ store-sops ys
  by (fastforce simp add: in-store-sops-conv)

lemma store-sops-append: store-sops (xs@ys) = store-sops xs $\cup$ store-sops ys
  by (force simp add: in-store-sops-conv)

locale valid-write-sops =
fixes ts::('p,'p store-buffer,bool,owns,rels) thread-config list
assumes valid-write-sops:
  $\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
  $[\![$i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)$]\!]$
  $\Longrightarrow$
  $\forall$sop $\in$ write-sops sb. valid-sop sop

locale valid-store-sops =
fixes ts::('p,'p store-buffer,bool,owns,rels) thread-config list
assumes valid-store-sops:
  $\bigwedge$i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
  $[\![$i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)$]\!]$
  $\Longrightarrow$
  $\forall$sop $\in$ store-sops is. valid-sop sop

locale valid-sops = valid-write-sops + valid-store-sops

The value stored in a non-volatile $\mathsf{Read_{sb}}$ in the store-buffer has to match the last value written to the same address in the store buffer or the memory content if there is no corresponding write in the store buffer. No volatile read may follow a volatile write. Volatile reads in the store buffer may refer to a stale value: e.g. imagine one writer and multiple readers**fun** reads-consistent:: bool $\Rightarrow$ owns $\Rightarrow$ memory $\Rightarrow$ 'p store-buffer $\Rightarrow$ bool
where
  reads-consistent pending-write $\mathcal{O}$ m [] = True
| reads-consistent pending-write $\mathcal{O}$ m (r#rs) =
   (case r of
      Read$_{\mathsf{sb}}$ volatile a t v $\Rightarrow$ ($\neg$ volatile $\longrightarrow$ (pending-write $\lor$ a $\in$ $\mathcal{O}$) $\longrightarrow$ v = m a) $\land$
                          reads-consistent pending-write $\mathcal{O}$ m rs
    | Write$_{\mathsf{sb}}$ volatile a sop v A L R W $\Rightarrow$
        (if volatile then
            outstanding-refs is-volatile-Read$_{\mathsf{sb}}$ rs = {} $\land$
            reads-consistent True ($\mathcal{O}$ $\cup$ A $-$ R) (m(a := v)) rs
         else reads-consistent pending-write $\mathcal{O}$ (m(a := v)) rs)
    | Ghost$_{\mathsf{sb}}$ A L R W $\Rightarrow$ reads-consistent pending-write ($\mathcal{O}$ $\cup$ A $-$ R) m rs
    | - $\Rightarrow$ reads-consistent pending-write $\mathcal{O}$ m rs
)

fun volatile-reads-consistent:: memory $\Rightarrow$ 'p store-buffer $\Rightarrow$ bool
where
  volatile-reads-consistent m [] = True
| volatile-reads-consistent m (r#rs) =
   (case r of
      Read$_{\mathsf{sb}}$ volatile a t v $\Rightarrow$ (volatile $\longrightarrow$ v = m a) $\land$ volatile-reads-consistent m rs
    | Write$_{\mathsf{sb}}$ volatile a sop v A L R W $\Rightarrow$ volatile-reads-consistent (m(a := v)) rs

| - ⇒ volatile-reads-consistent m rs
)

fun flush:: 'p store-buffer ⇒ memory ⇒ memory
where
flush [] m = m
| flush (r#rs) m =
(case r of
Write_{sb} volatile a - v - - - - ⇒ flush rs (m(a:=v))
| - ⇒ flush rs m)

lemma reads-consistent-pending-write-antimono:
$\bigwedge\mathcal{O}$ m. reads-consistent True $\mathcal{O}$ m sb $\Longrightarrow$ reads-consistent False $\mathcal{O}$ m sb
apply (induct sb)
apply simp
subgoal for a sb $\mathcal{O}$ m
by (case-tac a) auto
done

lemma reads-consistent-owns-antimono:
$\bigwedge\mathcal{O}$ $\mathcal{O}'$ pending-write m.
$\mathcal{O} \subseteq \mathcal{O}' \Longrightarrow$ reads-consistent pending-write $\mathcal{O}'$ m sb $\Longrightarrow$ reads-consistent pending-write $\mathcal{O}$ m sb
apply (induct sb)
apply simp
subgoal for a sb $\mathcal{O}$ $\mathcal{O}'$ pending-write m
apply (case-tac a)
apply (clarsimp split: if-split-asm)
subgoal for volatile a D f v A L R W
apply (drule-tac C=A **in** union-mono-aux)
apply (drule-tac C=R **in** set-minus-mono-aux)
apply blast
done
apply fastforce
apply fastforce
apply clarsimp
subgoal for A L R W
apply (drule-tac C=A **in** union-mono-aux)
apply (drule-tac C=R **in** set-minus-mono-aux)
apply blast
done
done
done

lemma acquired-reads-mono': x ∈ acquired-reads b xs A $\Longrightarrow$ acquired-reads b xs B = {} $\Longrightarrow$ A $\subseteq$ B $\Longrightarrow$ False
apply (drule acquired-reads-mono-in [**where** B=B])
apply auto
done

lemma reads-consistent-append:
$\bigwedge$m pending-write $\mathcal{O}$. reads-consistent pending-write $\mathcal{O}$ m (xs@ys) =
(reads-consistent pending-write $\mathcal{O}$ m xs $\wedge$
reads-consistent (pending-write $\vee$ outstanding-refs is-volatile-Write$_{sb}$ xs $\neq$ {})
(acquired True xs $\mathcal{O}$) (flush xs m) ys $\wedge$
(outstanding-refs is-volatile-Write$_{sb}$ xs $\neq$ {}
$\longrightarrow$ outstanding-refs is-volatile-Read$_{sb}$ ys = {}))
apply (induct xs)
apply clarsimp
subgoal for a xs m pending-write $\mathcal{O}$
apply (case-tac a)
apply (auto simp add: outstanding-refs-append acquired-reads-append
dest: acquired-reads-mono-in acquired-pending-write-mono-in acquired-reads-mono′ acquired-mono-in)
done
done

lemma reads-consistent-mem-eq-on-non-volatile-reads:
assumes mem-eq: $\forall$a $\in$ A. m′ a = m a
assumes subset: outstanding-refs (is-non-volatile-Read$_{sb}$) sb $\subseteq$ A
— We could be even more restrictive here, only the non volatile reads that are not buffered in *sb* have to be the same.
assumes consis-m: reads-consistent pending-write $\mathcal{O}$ m sb
shows reads-consistent pending-write $\mathcal{O}$ m′ sb
using mem-eq subset consis-m
proof (induct sb arbitrary: m′ m pending-write $\mathcal{O}$)
case Nil **thus** ?case **by** simp
next
case (Cons r sb)
note mem-eq = ‹$\forall$a $\in$ A. m′ a = m a›
note subset = ‹outstanding-refs (is-non-volatile-Read$_{sb}$) (r#sb) $\subseteq$ A›
note consis-m = ‹reads-consistent pending-write $\mathcal{O}$ m (r#sb)›

from subset **have** subset′: outstanding-refs is-non-volatile-Read$_{sb}$ sb $\subseteq$ A
by (auto simp add: Write$_{sb}$)
show ?case
proof (cases r)
case (Write$_{sb}$ volatile a sop v A′ L R W)
from mem-eq
have mem-eq′:
$\forall$a′ $\in$ A. (m′(a:=v)) a′ = (m(a:=v)) a′
by (auto)
show ?thesis
proof (cases volatile)
case True
from consis-m **obtain**
consis′: reads-consistent True ($\mathcal{O}$ $\cup$ A′ − R) (m(a := v)) sb **and**
no-volatile-Read$_{sb}$: outstanding-refs is-volatile-Read$_{sb}$ sb = {}

by (simp add: $\text{Write}_{\mathsf{sb}}$ True)

from Cons.hyps [OF mem-eq′ subset′ consis′]
have reads-consistent True ($\mathcal{O} \cup A' - R$) ($m'(a := v)$) sb.
with no-volatile-$\text{Read}_{\mathsf{sb}}$
show ?thesis
by (simp add: $\text{Write}_{\mathsf{sb}}$ True)
next
case False
from consis-m **obtain** consis′: reads-consistent pending-write $\mathcal{O}$ ($m(a := v)$) sb
by (simp add: $\text{Write}_{\mathsf{sb}}$ False)
from Cons.hyps [OF mem-eq′ subset′ consis′]
have reads-consistent pending-write $\mathcal{O}$ ($m'(a := v)$) sb.
then
show ?thesis
by (simp add: $\text{Write}_{\mathsf{sb}}$ False)
qed
next
case ($\text{Read}_{\mathsf{sb}}$ volatile a t v)
from mem-eq
have mem-eq′:
$\forall a' \in A.\ m'\ a' = m\ a'$
by (auto)
show ?thesis
proof (cases volatile)
case True
from consis-m **obtain**
consis′: reads-consistent pending-write $\mathcal{O}$ m sb
by (simp add: $\text{Read}_{\mathsf{sb}}$ True)
from Cons.hyps [OF mem-eq′ subset′ consis′]
show ?thesis
by (simp add: $\text{Read}_{\mathsf{sb}}$ True)
next
case False
from consis-m **obtain**
consis′: reads-consistent pending-write $\mathcal{O}$ m sb **and** v: (pending-write $\vee$ $a \in \mathcal{O}$) $\longrightarrow$ v=m
a
by (simp add: $\text{Read}_{\mathsf{sb}}$ False)
from mem-eq subset $\text{Read}_{\mathsf{sb}}$ **have** $m'\ a = m\ a$
by (auto simp add: False)
with Cons.hyps [OF mem-eq′ subset′ consis′] v
show ?thesis
by (simp add: $\text{Read}_{\mathsf{sb}}$ False)
qed
next
case $\text{Prog}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** auto
next
case $\text{Ghost}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** auto
qed
qed

```
lemma volatile-reads-consistent-mem-eq-on-volatile-reads:
  assumes mem-eq: ∀a ∈ A. m′ a = m a
  assumes subset: outstanding-refs (is-volatile-Read_sb) sb ⊆ A
  — We could be even more restrictive here, only the non volatile reads that are not
buffered in sb have to be the same.
  assumes consis-m: volatile-reads-consistent m sb
  shows volatile-reads-consistent m′ sb
using mem-eq subset consis-m
proof (induct sb arbitrary: m′ m)
  case Nil thus ?case by simp
next
  case (Cons r sb)
  note mem-eq = ‹∀a ∈ A. m′ a = m a›
  note subset = ‹outstanding-refs (is-volatile-Read_sb) (r#sb) ⊆ A›
  note consis-m = ‹volatile-reads-consistent m (r#sb)›

  from subset have subset′: outstanding-refs is-volatile-Read_sb sb ⊆ A
    by (auto simp add: Write_sb)
  show ?case
  proof (cases r)
    case (Write_sb volatile a sop v A′ L R W)
    from mem-eq
    have mem-eq′:
      ∀a′ ∈ A. (m′(a:=v)) a′ = (m(a:=v)) a′
      by (auto)
    show ?thesis
    proof (cases volatile)
      case True
      from consis-m obtain
 consis′: volatile-reads-consistent (m(a := v)) sb
 by (simp add: Write_sb True)

      from Cons.hyps [OF mem-eq′ subset′ consis′]
      have volatile-reads-consistent (m′(a := v)) sb.
      then
      show ?thesis
 by (simp add: Write_sb True)
    next
      case False
      from consis-m obtain consis′: volatile-reads-consistent (m(a := v)) sb
 by (simp add: Write_sb False)
      from Cons.hyps [OF mem-eq′ subset′ consis′]
      have volatile-reads-consistent (m′(a := v)) sb.
      then
      show ?thesis
 by (simp add: Write_sb False)
    qed
```

next
case ($Read_{sb}$ volatile a t v)
from mem-eq
have mem-eq′:
∀ a′ ∈ A. m′ a′ = m a′
by (auto)
show ?thesis
proof (cases volatile)
case False
from consis-m **obtain**
consis′: volatile-reads-consistent m sb
by (simp add: $Read_{sb}$ False)
from Cons.hyps [OF mem-eq′ subset′ consis′]
show ?thesis
by (simp add: $Read_{sb}$ False)
next
case True
from consis-m **obtain**
consis′: volatile-reads-consistent m sb **and** v: v=m a
by (simp add: $Read_{sb}$ True)
from mem-eq subset $Read_{sb}$ v **have** v = m′ a
by (auto simp add: True)
with Cons.hyps [OF mem-eq′ subset′ consis′]
show ?thesis
by (simp add: $Read_{sb}$ True)
qed
next
case $Prog_{sb}$ **with** Cons **show** ?thesis **by** auto
next
case $Ghost_{sb}$ **with** Cons **show** ?thesis **by** auto
qed
qed

locale valid-reads =
fixes m::memory **and** ts::(′p, ′p store-buffer,bool,owns,rels) thread-config list
assumes valid-reads: ⋀i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟧ ⟹
reads-consistent False $\mathcal{O}$ m sb

lemma valid-reads-Cons: valid-reads m (t#ts) =
(let (-,-,-,sb,-,$\mathcal{O}$,-) = t in reads-consistent False $\mathcal{O}$ m sb ∧ valid-reads m ts)
apply (auto simp add: valid-reads-def)
subgoal for p′ is′ j′ sb′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$ i p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$
apply (case-tac i)
apply auto
done
done

$Read_{sb}$s and writes have in the store-buffer have to conform to the valuation of temporaries.**context** program
begin

fun history-consistent:: tmps $\Rightarrow$ $'$p $\Rightarrow$ $'$p store-buffer $\Rightarrow$ bool
where
  history-consistent j p [] = True
| history-consistent j p (r#rs) =
    (case r of
      $\text{Read}_{\mathsf{sb}}$ vol a t v $\Rightarrow$
        (case j t of Some v$'$ $\Rightarrow$ v=v$'$ $\wedge$ history-consistent j p rs | - $\Rightarrow$ False)
    | $\text{Write}_{\mathsf{sb}}$ vol a (D,f) v - - - - $\Rightarrow$
        D $\subseteq$ dom j $\wedge$ f j = v $\wedge$ D $\cap$ read-tmps rs = {} $\wedge$ history-consistent j p rs
    | $\text{Prog}_{\mathsf{sb}}$ p_1 p_2 is $\Rightarrow$ p_1=p $\wedge$
                    j|`(dom j $-$ read-tmps rs)$\vdash$ p_1 $\rightarrow_{\mathsf{p}}$ (p_2,is) $\wedge$
                    history-consistent j p_2 rs
    | - $\Rightarrow$ history-consistent j p rs)
end

fun hd-prog:: $'$p $\Rightarrow$ $'$p store-buffer $\Rightarrow$ $'$p
where
  hd-prog p [] = p
| hd-prog p (i#is) = (case i of
      $\text{Prog}_{\mathsf{sb}}$ p$'$ - - $\Rightarrow$ p$'$
    | - $\Rightarrow$ hd-prog p is)

fun last-prog:: $'$p $\Rightarrow$ $'$p store-buffer $\Rightarrow$ $'$p
where
  last-prog p [] = p
| last-prog p (i#is) = (case i of
      $\text{Prog}_{\mathsf{sb}}$ - p$'$ - $\Rightarrow$ last-prog p$'$ is
    | - $\Rightarrow$ last-prog p is)

locale valid-history = program +
constrains
  program-step :: tmps $\Rightarrow$ $'$p $\Rightarrow$ $'$p $\times$ instrs $\Rightarrow$ bool
fixes ts::($'$p,$'$p store-buffer,bool,owns,rels) thread-config list
assumes valid-history: $\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
    $[\![$i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $]\!]$ $\Longrightarrow$
      program.history-consistent program-step j (hd-prog p sb) sb

fun data-dependency-consistent-instrs:: addr set $\Rightarrow$ instrs $\Rightarrow$ bool
where
  data-dependency-consistent-instrs T [] = True
| data-dependency-consistent-instrs T (i#is) =
    (case i of
      Write volatile a (D,f) - - - - $\Rightarrow$ D $\subseteq$ T $\wedge$ D $\cap$ load-tmps is = {} $\wedge$ data-dependency-consistent-instrs T is
    | RMW a t (D,f) cond ret - - - - $\Rightarrow$ D $\subseteq$ T $\wedge$ D $\cap$ load-tmps is = {} $\wedge$ data-dependency-consistent-instrs (insert t T) is
    | Read - - t $\Rightarrow$ data-dependency-consistent-instrs (insert t T) is
    | - $\Rightarrow$ data-dependency-consistent-instrs T is)

lemma data-dependency-consistent-mono:

$\bigwedge$ T T′. ⟦data-dependency-consistent-instrs T is; T ⊆ T′⟧ ⟹ data-dependency-consistent-instrs T′ is
apply (induct is)
apply clarsimp
subgoal for a is T T′
apply (case-tac a)
apply clarsimp
  subgoal for volatile a′ t
  apply (drule-tac a=t **in** insert-mono)
  apply clarsimp
  done
apply fastforce
apply clarsimp
  subgoal for a′ t D f cond ret A L R W
  apply (frule-tac a=t **in** insert-mono)
  apply fastforce
  done
apply fastforce
apply fastforce
done
done

lemma data-dependency-consistent-instrs-append:
$\bigwedge$ys T . data-dependency-consistent-instrs T (xs@ys) =
  (data-dependency-consistent-instrs T xs ∧
  data-dependency-consistent-instrs (T ∪ load-tmps xs) ys ∧
  load-tmps ys ∩ $\bigcup$(fst ‘ store-sops xs) = {})
apply (induct xs)
apply (auto split: instr.splits simp add: load-tmps-append intro: data-dependency-consistent-mono)
done

locale valid-data-dependency =
fixes ts::(′p,′p store-buffer,bool,owns,rels) thread-config list
assumes data-dependency-consistent-instrs:
$\bigwedge$i p is $\mathcal{O}$ $\mathcal{D}$ j sb.
  ⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟧ ⟹
  data-dependency-consistent-instrs (dom j) is
assumes load-tmps-write-tmps-distinct:
$\bigwedge$i p is $\mathcal{O}$ $\mathcal{D}$ j sb.
  ⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟧ ⟹
  load-tmps is ∩ $\bigcup$(fst ‘ write-sops sb) = {}

locale load-tmps-fresh =
fixes ts::(′p, ′p store-buffer,bool,owns,rels) thread-config list
assumes load-tmps-fresh:
$\bigwedge$i p is $\mathcal{O}$ $\mathcal{D}$ j sb.
  ⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟧ ⟹
  load-tmps is ∩ dom j = {}

fun acquired-by-instrs :: instrs ⇒ addr set ⇒ addr set
where
  acquired-by-instrs [] A = A
| acquired-by-instrs (i#is) A =
   (case i of
      Read - - - ⇒ acquired-by-instrs is A
    | Write volatile - - A′ L R W ⇒ acquired-by-instrs is (if volatile then (A ∪ A′ − R) else A)
    | RMW a t sop cond ret A′ L R W ⇒ acquired-by-instrs is {}
    | Fence ⇒ acquired-by-instrs is {}
    | Ghost A′ L R W ⇒ acquired-by-instrs is (A ∪ A′ − R))

fun acquired-loads :: bool ⇒ instrs ⇒ addr set ⇒ addr set
where
  acquired-loads pending-write [] A = {}
| acquired-loads pending-write (i#is) A =
   (case i of
      Read volatile a - ⇒ (if pending-write ∧ ¬ volatile ∧ a ∈ A
                          then insert a (acquired-loads pending-write is A)
                          else acquired-loads pending-write is A)
   | Write volatile - - A′ L R W ⇒ (if volatile then acquired-loads True is (if pending-write then (A ∪ A′ − R) else {})
                          else acquired-loads pending-write is A)
    | RMW a t sop cond ret A′ L R W ⇒ acquired-loads pending-write is {}
    | Fence ⇒ acquired-loads pending-write is {}
    | Ghost A′ L R W ⇒ acquired-loads pending-write is (A ∪ A′ − R))

lemma acquired-by-instrs-mono:
  ⋀ A B. A ⊆ B ⟹ acquired-by-instrs is A ⊆ acquired-by-instrs is B
apply (induct is)
apply simp
subgoal for a is A B
apply (case-tac a)
apply clarsimp
apply clarsimp
    subgoal for volatile a′ D f A′ L R W x
    apply (drule-tac C=A′ **in** union-mono-aux)
    apply (drule-tac C=R **in** set-minus-mono-aux)
    apply blast
    done
apply clarsimp
apply clarsimp
apply clarsimp
subgoal for A′ L R W x
apply (drule-tac C=A′ **in** union-mono-aux)
apply (drule-tac C=R **in** set-minus-mono-aux)
apply blast
done
done

done

lemma acquired-by-instrs-mono-in:
  assumes x-in: $x \in$ acquired-by-instrs is A
  assumes sub: $A \subseteq B$
  shows $x \in$ acquired-by-instrs is B
using acquired-by-instrs-mono [OF sub, of is] x-in
by blast

locale enough-flushs =
fixes ts::('p,'p store-buffer,bool,owns,rels) thread-config list
assumes clean-no-outstanding-volatile-Write$_{sb}$:
  $\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
    $[\![$i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);$\neg$ $\mathcal{D}$$]\!] \Longrightarrow$
      (outstanding-refs is-volatile-Write$_{sb}$ sb = {})

fun prog-instrs:: 'p store-buffer $\Rightarrow$ instrs
where
  prog-instrs [] = []
 |prog-instrs (i#is) = (case i of
    Prog$_{sb}$ - - is' $\Rightarrow$ is' @ prog-instrs is
   | - $\Rightarrow$ prog-instrs is)

fun instrs:: 'p store-buffer $\Rightarrow$ instrs
where
  instrs [] = []
| instrs (i#is) = (case i of
    Write$_{sb}$ volatile a sop v A L R W $\Rightarrow$ Write volatile a sop A L R W# instrs is
   | Read$_{sb}$ volatile a t v $\Rightarrow$ Read volatile a t # instrs is
   | Ghost$_{sb}$ A L R W $\Rightarrow$ Ghost A L R W# instrs is
   | - $\Rightarrow$ instrs is)

locale causal-program-history =
fixes is$_{sb}$ **and** sb
assumes causal-program-history:
  $\bigwedge$sb$_1$ sb$_2$. sb=sb$_1$@sb$_2$ $\Longrightarrow$ $\exists$is. instrs sb$_2$ @ is$_{sb}$ = is @ prog-instrs sb$_2$

lemma causal-program-history-empty [simp]: causal-program-history is []
  by (rule causal-program-history.intro) simp

lemma causal-program-history-suffix:
  causal-program-history is$_{sb}$ (sb@sb') $\Longrightarrow$ causal-program-history is$_{sb}$ sb'
  by (auto simp add: causal-program-history-def)

locale valid-program-history =
fixes ts::('p,'p store-buffer,bool,owns,rels) thread-config list
assumes valid-program-history:
  $\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
    $[\![$i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $]\!] \Longrightarrow$

causal-program-history is sb

assumes valid-last-prog:
$\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟧ $\Longrightarrow$
last-prog p sb = p

lemma (**in** valid-program-history) valid-program-history-nth-update:
⟦i < length ts; causal-program-history is sb; last-prog p sb = p⟧
$\Longrightarrow$
valid-program-history (ts [i:=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])
by (rule valid-program-history.intro)
(auto dest: valid-program-history valid-last-prog
simp add: nth-list-update split: if-split-asm)

lemma (**in** outstanding-non-volatile-refs-owned-or-read-only)
outstanding-non-volatile-refs-owned-instructions-read-value-independent:
$\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟧ $\Longrightarrow$
outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}$ (ts[i := (p′,is′,j′,sb,$\mathcal{D}'$,$\mathcal{O}$,$\mathcal{R}'$)])
by (unfold-locales)
(auto dest: outstanding-non-volatile-refs-owned-or-read-only
simp add: nth-list-update split: if-split-asm)

lemma (**in** outstanding-non-volatile-refs-owned-or-read-only)
outstanding-non-volatile-refs-owned-or-read-only-nth-update:
$\bigwedge$i is $\mathcal{O}$ $\mathcal{D}$ $\mathcal{R}$ j sb.
⟦i < length ts; non-volatile-owned-or-read-only False $\mathcal{S}$ $\mathcal{O}$ sb⟧ $\Longrightarrow$
outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}$ (ts[i := (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])
by (unfold-locales)
(auto dest: outstanding-non-volatile-refs-owned-or-read-only
simp add: nth-list-update split: if-split-asm)

lemma (**in** outstanding-volatile-writes-unowned-by-others)
outstanding-volatile-writes-unowned-by-others-instructions-read-value-independent:
$\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟧ $\Longrightarrow$
outstanding-volatile-writes-unowned-by-others (ts[i := (p′,is′,j′,sb,$\mathcal{D}'$,$\mathcal{O}$,$\mathcal{R}'$)])
by (unfold-locales)
(auto dest: outstanding-volatile-writes-unowned-by-others
simp add: nth-list-update split: if-split-asm)

lemma (**in** read-only-reads-unowned)
read-only-unowned-instructions-read-value-independent:
$\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟧ $\Longrightarrow$
read-only-reads-unowned (ts[i := (p′,is′,j′,sb,$\mathcal{D}'$,$\mathcal{O}$,$\mathcal{R}'$)])
by (unfold-locales)
(auto dest: read-only-reads-unowned
simp add: nth-list-update split: if-split-asm)

lemma $\text{Write}_{\mathsf{sb}}$-in-outstanding-refs:
$\text{Write}_{\mathsf{sb}}$ True a sop v A L R W $\in$ set xs $\Longrightarrow$ a $\in$ outstanding-refs is-volatile-$\text{Write}_{\mathsf{sb}}$ xs
by (induct xs) (auto split:memref.splits)

lemma (**in** outstanding-volatile-writes-unowned-by-others)
outstanding-volatile-writes-unowned-by-others-store-buffer:
$\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
$[\![$i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
outstanding-refs is-volatile-$\text{Write}_{\mathsf{sb}}$ sb$'$ $\subseteq$ outstanding-refs is-volatile-$\text{Write}_{\mathsf{sb}}$ sb;
all-acquired sb$'$ $\subseteq$ all-acquired sb$]\!]$ $\Longrightarrow$
outstanding-volatile-writes-unowned-by-others (ts[i := (p$'$,is$'$,j$'$,sb$'$,$\mathcal{D}'$,$\mathcal{O}$,$\mathcal{R}'$)])
apply (unfold-locales)
apply (fastforce dest: outstanding-volatile-writes-unowned-by-others
simp add: nth-list-update split: if-split-asm)
done

lemma (**in** ownership-distinct)
ownership-distinct-instructions-read-value-store-buffer-independent:
$\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
$[\![$i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
all-acquired sb$'$ $\subseteq$ all-acquired sb$]\!]$ $\Longrightarrow$
ownership-distinct (ts[i := (p$'$,is$'$,j$'$,sb$'$,$\mathcal{D}'$,$\mathcal{O}$,$\mathcal{R}'$)])
by (unfold-locales)
(auto dest: ownership-distinct
simp add: nth-list-update split: if-split-asm)

lemma (**in** ownership-distinct)
ownership-distinct-nth-update:
$\bigwedge$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ xs sb.
$[\![$i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
$\forall$j < length ts. i$\neq$j $\longrightarrow$ (let (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$) = ts!j
in ($\mathcal{O}'$ $\cup$ all-acquired sb$'$) $\cap$ ($\mathcal{O}_j$ $\cup$ all-acquired sb_j) ={}) $]\!]$ $\Longrightarrow$
ownership-distinct (ts[i := (p$'$,is$'$,j$'$,sb$'$,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)])
apply (unfold-locales)
apply (clarsimp simp add: nth-list-update split: if-split-asm)
apply (force dest: ownership-distinct simp add: Let-def)
apply (fastforce dest: ownership-distinct simp add: Let-def)
apply (fastforce dest: ownership-distinct simp add: Let-def)
done

lemma (**in** valid-write-sops) valid-write-sops-nth-update:
$[\![$i < length ts; $\forall$sop $\in$ write-sops sb. valid-sop sop$]\!]$ $\Longrightarrow$
valid-write-sops (ts[i := (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])

by (unfold valid-write-sops-def)
(auto dest: valid-write-sops simp add: nth-list-update split: if-split-asm)

lemma (**in** valid-store-sops) valid-store-sops-nth-update:
$[\![$i < length ts; $\forall$sop $\in$ store-sops is. valid-sop sop$]\!] \Longrightarrow$
valid-store-sops (ts[i := (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])
by (unfold valid-store-sops-def)
(auto dest: valid-store-sops simp add: nth-list-update split: if-split-asm)

lemma (**in** valid-sops) valid-sops-nth-update:
$[\![$i < length ts; $\forall$sop $\in$ write-sops sb. valid-sop sop;
$\forall$sop $\in$ store-sops is. valid-sop sop$]\!] \Longrightarrow$
valid-sops (ts[i := (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])
by (unfold valid-sops-def valid-write-sops-def valid-store-sops-def)
(auto dest: valid-write-sops valid-store-sops
simp add: nth-list-update split: if-split-asm)

lemma (**in** valid-data-dependency) valid-data-dependency-nth-update:
$[\![$i < length ts; data-dependency-consistent-instrs (dom j) is;
load-tmps is $\cap \bigcup$(fst ‘ write-sops sb) = {}$]\!] \Longrightarrow$
valid-data-dependency (ts[i := (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])
by (unfold valid-data-dependency-def)
(force dest: data-dependency-consistent-instrs load-tmps-write-tmps-distinct
simp add: nth-list-update split: if-split-asm)

lemma (**in** enough-flushs) enough-flushs-nth-update:
$[\![$i < length ts;
$\neg$ $\mathcal{D}$ $\longrightarrow$ (outstanding-refs is-volatile-Write$_{\mathsf{sb}}$ sb = {})
$]\!] \Longrightarrow$
enough-flushs (ts[i := (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])

apply (unfold-locales)
apply (force simp add: nth-list-update split: if-split-asm dest: clean-no-outstanding-volatile-Write$_{\mathsf{sb}}$)
done

lemma (**in** outstanding-non-volatile-writes-unshared)
outstanding-non-volatile-writes-unshared-nth-update:
$[\![$i < length ts; non-volatile-writes-unshared $\mathcal{S}$ sb$]\!] \Longrightarrow$
outstanding-non-volatile-writes-unshared $\mathcal{S}$ (ts[i := (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])
by (unfold-locales)
(auto dest: outstanding-non-volatile-writes-unshared
simp add: nth-list-update split: if-split-asm)

lemma (**in** sharing-consis)
sharing-consis-nth-update:
$[\![$i < length ts; sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb$]\!] \Longrightarrow$
sharing-consis $\mathcal{S}$ (ts[i := (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])
by (unfold-locales)

(auto dest: sharing-consis
  simp add: nth-list-update split: if-split-asm)

lemma (**in** no-outstanding-write-to-read-only-memory)
no-outstanding-write-to-read-only-memory-nth-update:
$\llbracket$i < length ts; no-write-to-read-only-memory $\mathcal{S}$ sb$\rrbracket \Longrightarrow$
no-outstanding-write-to-read-only-memory $\mathcal{S}$ (ts[i := (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])
by (unfold-locales)
(auto dest: no-outstanding-write-to-read-only-memory
  simp add: nth-list-update split: if-split-asm)

lemma in-Union-image-nth-conv: a $\in \bigcup$ (f ‘ set xs) $\Longrightarrow \exists$i. i < length xs $\wedge$ a $\in$ f (xs!i)
by (auto simp add: in-set-conv-nth)

lemma in-Inter-image-nth-conv: a $\in \bigcap$ (f ‘ set xs) = ($\forall$i < length xs. a $\in$ f (xs!i))
by (force simp add: in-set-conv-nth)

lemma release-ownership-nth-update:
assumes R-subset: R $\subseteq \mathcal{O}$
shows $\bigwedge$i. $\llbracket$i < length ts; ts!i = (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
ownership-distinct ts$\rrbracket$
$\Longrightarrow \bigcup$ ((λ(-,-,-,-,-,$\mathcal{O}$,-). $\mathcal{O}$) ‘ set (ts[i:=(p',is',xs',sb',$\mathcal{D}'$,$\mathcal{O}$ − R,$\mathcal{R}'$)]))
= (($\bigcup$ ((λ(-,-,-,-,-,$\mathcal{O}$,-). $\mathcal{O}$) ‘ set ts)) − R)
proof (induct ts)
case Nil **thus** ?case **by** simp
next
case (Cons t ts)
note i-bound = ‹i < length (t # ts)›
note ith = ‹(t # ts) ! i = (p,is,xs, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)›
note dist = ‹ownership-distinct (t#ts)›
then interpret ownership-distinct t#ts**.**
from dist
have dist': ownership-distinct ts
by (rule ownership-distinct-tl)
show ?case
proof (cases i)
case 0
from ith 0 **have** t: t = (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by simp
have R $\cap$ ($\bigcup$ ((λ(-,-,-,-,-,$\mathcal{O}$,-). $\mathcal{O}$) ‘ set ts)) = {}
proof −
{
fix x
assume x-R: x $\in$ R
assume x-ls: x $\in$ ($\bigcup$ ((λ(-,-,-,-,-,$\mathcal{O}$,-). $\mathcal{O}$) ‘ set ts))
then obtain j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs_j sb_j **where**

j-bound: j < length ts **and**
jth: ts!j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$) **and**
x-in: x ∈ $\mathcal{O}_j$
by (fastforce simp add: in-set-conv-nth)
from j-bound jth 0
have ($\mathcal{O}$ ∪ all-acquired sb) ∩ ($\mathcal{O}_j$ ∪ all-acquired sb_j)= {}
apply −
apply (rule ownership-distinct [OF i-bound - - ith, of Suc j])
apply clarsimp+
apply blast
done

with x-R R-subset x-in **have** False
by auto
}
thus ?thesis
by blast
qed
then
show ?thesis
by (auto simp add: 0 t)
next
case (Suc n)
obtain p_l is_l $\mathcal{O}_l$ $\mathcal{R}_l$ $\mathcal{D}_l$ xs_l sb_l **where** t: t = (p_l,is_l,xs_l,sb_l,$\mathcal{D}_l$,$\mathcal{O}_l$,$\mathcal{R}_l$)
by (cases t)

have n-bound: n < length ts
using i-bound **by** (simp add: Suc)
have nth: ts!n = (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
using ith **by** (simp add: Suc)

have R ∩ ($\mathcal{O}_l$ ∪ all-acquired sb_l) = {}
proof −
{
fix x
assume x-R: x ∈ R
assume x-owns$_l$: x ∈ ($\mathcal{O}_l$ ∪ all-acquired sb_l)
from t
have ($\mathcal{O}$ ∪ all-acquired sb) ∩ ($\mathcal{O}_l$ ∪ all-acquired sb_l)= {}
apply −
apply (rule ownership-distinct [OF i-bound - - ith, of 0])
apply (auto simp add: Suc)
done
with x-owns$_l$ x-R R-subset **have** False
by auto
}
thus ?thesis
by blast
qed
with Cons.hyps [OF n-bound nth dist′]

```
    show ?thesis
      by (auto simp add: Suc t)
  qed
qed

lemma acquire-ownership-nth-update:
  shows ⋀i. ⟦i < length ts; ts!i = (p,is,xs,sb,𝒟,𝒪,ℛ)⟧
   ⟹ ⋃ ((λ(-,-,-,-,-,𝒪,-). 𝒪) ‘ set (ts[i:=(p′,is′,xs′,sb′,𝒟′,𝒪 ∪ A,ℛ′)]))
        = ((⋃ ((λ(-,-,-,-,-,𝒪,-). 𝒪) ‘ set ts)) ∪ A )
proof (induct ts)
  case Nil thus ?case by simp
next
  case (Cons t ts)
  note i-bound = ‹i < length (t # ts)›
  note ith = ‹(t # ts) ! i = (p,is, xs, sb, 𝒟, 𝒪, ℛ)›
  show ?case
  proof (cases i)
    case 0
    from ith 0 have t: t = (p,is,xs,sb,𝒟,𝒪,ℛ)
      by simp
    show ?thesis
      by (auto simp add: 0 t)
  next
    case (Suc n)
    obtain pₗ isₗ 𝒪ₗ ℛₗ 𝒟ₗ xsₗ sbₗ where t: t = (pₗ,isₗ,xsₗ,sbₗ,𝒟ₗ,𝒪ₗ,ℛₗ)
      by (cases t)

    have n-bound: n < length ts
      using i-bound by (simp add: Suc)
    have nth: ts!n = (p,is,xs,sb,𝒟,𝒪,ℛ)
      using ith by (simp add: Suc)
    from Cons.hyps [OF n-bound nth]
    show ?thesis
      by (auto simp add: Suc t)
  qed
qed

lemma acquire-release-ownership-nth-update:
  assumes R-subset: R ⊆ 𝒪
  shows ⋀i. ⟦i < length ts; ts!i = (p,is,xs,sb,𝒟,𝒪,ℛ);
          ownership-distinct ts⟧
   ⟹ ⋃ ((λ(-,-,-,-,-,𝒪,-). 𝒪) ‘ set (ts[i:=(p′,is′,xs′,sb′,𝒟′,𝒪 ∪ A − R,ℛ′)]))
        = ((⋃ ((λ(-,-,-,-,-,𝒪,-). 𝒪) ‘ set ts)) ∪ A − R )
proof (induct ts)
  case Nil thus ?case by simp
next
  case (Cons t ts)
  note i-bound = ‹i < length (t # ts)›
  note ith = ‹(t # ts) ! i = (p,is, xs, sb,𝒟, 𝒪,ℛ)›
  note dist = ‹ownership-distinct (t#ts)›
```

then interpret ownership-distinct t#ts**.**
from dist
have dist$'$: ownership-distinct ts
by (rule ownership-distinct-tl)
show ?case
proof (cases i)
case 0
from ith 0 **have** t: t = (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by simp
have R $\cap$ ($\bigcup$ ((λ(-,-,-,-,-,$\mathcal{O}$,-). $\mathcal{O}$) ' set ts)) = {}
proof $-$
{
fix x
assume x-R: x $\in$ R
assume x-ls: x $\in$ ($\bigcup$ ((λ(-,-,-,-,-,$\mathcal{O}$,-). $\mathcal{O}$) ' set ts))
then obtain j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs_j sb_j **where**
j-bound: j < length ts **and**
jth: ts!j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$) **and**
x-in: x $\in$ $\mathcal{O}_j$
by (fastforce simp add: in-set-conv-nth)
from j-bound jth 0
have ($\mathcal{O}$ $\cup$ all-acquired sb) $\cap$ ($\mathcal{O}_j$ $\cup$ all-acquired sb_j)= {}
apply $-$
apply (rule ownership-distinct [OF i-bound - - ith, of Suc j])
apply clarsimp+
apply blast
done

with x-R R-subset x-in **have** False
by auto
}
thus ?thesis
by blast
qed
then
show ?thesis
by (auto simp add: 0 t)
next
case (Suc n)
obtain p_l is_l $\mathcal{O}_l$ $\mathcal{R}_l$ $\mathcal{D}_l$ xs_l sb_l **where** t: t = (p_l,is_l,xs_l,sb_l,$\mathcal{D}_l$,$\mathcal{O}_l$,$\mathcal{R}_l$)
by (cases t)

have n-bound: n < length ts
using i-bound **by** (simp add: Suc)
have nth: ts!n = (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
using ith **by** (simp add: Suc)

have R $\cap$ ($\mathcal{O}_l$ $\cup$ all-acquired sb_l) = {}
proof $-$
{

fix x
assume x-R: x ∈ R
assume x-owns$_l$: x ∈ ($\mathcal{O}_l$ ∪ all-acquired sb$_l$)
from t
have ($\mathcal{O}$ ∪ all-acquired sb) ∩ ($\mathcal{O}_l$ ∪ all-acquired sb$_l$)= {}
apply −
apply (rule ownership-distinct [OF i-bound - - ith, of 0])
apply (auto simp add: Suc)
done
with x-owns$_l$ x-R R-subset **have** False
by auto
}
thus ?thesis
by blast
qed
with Cons.hyps [OF n-bound nth dist′]
show ?thesis
by (auto simp add: Suc t)
qed
qed

lemma (**in** valid-history) valid-history-nth-update:
⟦i < length ts; history-consistent j (hd-prog p sb) sb ⟧ ⟹
valid-history program-step (ts[i := (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])
by (unfold-locales)
(auto dest: valid-history simp add: nth-list-update split: if-split-asm)

lemma (**in** valid-reads) valid-reads-nth-update:
⟦i < length ts; reads-consistent False $\mathcal{O}$ m sb ⟧ ⟹
valid-reads m (ts[i := (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])
by (unfold-locales)
(auto dest: valid-reads simp add: nth-list-update split: if-split-asm)

lemma (**in** load-tmps-distinct) load-tmps-distinct-nth-update:
⟦i < length ts; distinct-load-tmps is⟧ ⟹
load-tmps-distinct (ts[i := (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])
by (unfold-locales)
(auto dest: load-tmps-distinct simp add: nth-list-update split: if-split-asm)

lemma (**in** read-tmps-distinct) read-tmps-distinct-nth-update:
⟦i < length ts; distinct-read-tmps sb⟧ ⟹
read-tmps-distinct (ts[i := (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])
by (unfold-locales)
(auto dest: read-tmps-distinct simp add: nth-list-update split: if-split-asm)

lemma (**in** load-tmps-read-tmps-distinct) load-tmps-read-tmps-distinct-nth-update:
⟦i < length ts; load-tmps is ∩ read-tmps sb = {}⟧ ⟹
load-tmps-read-tmps-distinct (ts[i := (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])

by (unfold-locales)
(auto dest: load-tmps-read-tmps-distinct simp add: nth-list-update split: if-split-asm)

lemma (**in** load-tmps-fresh) load-tmps-fresh-nth-update:
$[\![$i < length ts;
load-tmps is $\cap$ dom j = {}$]\!] \Longrightarrow$
load-tmps-fresh (ts[i := (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])
by (unfold-locales)
(fastforce dest: load-tmps-fresh
simp add: nth-list-update split: if-split-asm)

fun flush-all-until-volatile-write::
('p,'p store-buffer,'dirty,'owns,'rels) thread-config list $\Rightarrow$ memory $\Rightarrow$ memory
where
flush-all-until-volatile-write [] m = m
| flush-all-until-volatile-write ((-, -, -, sb,-, -)#ts) m =
flush-all-until-volatile-write ts (flush (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) m)

fun share-all-until-volatile-write::
('p,'p store-buffer,'dirty,'owns,'rels) thread-config list $\Rightarrow$ shared $\Rightarrow$ shared
where
share-all-until-volatile-write [] S = S
| share-all-until-volatile-write ((-, -, -, sb,-,-)#ts) S =
share-all-until-volatile-write ts (share (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) S)

lemma takeWhile-dropWhile-real-prefix:
$[\![$x $\in$ set xs; $\neg$ P x$]\!] \Longrightarrow \exists$y ys. xs=takeWhile P xs @ y#ys $\wedge$ $\neg$ P y $\wedge$ dropWhile P xs = y#ys
by (induct xs) auto

lemma buffered-val-witness: buffered-val sb a = Some v $\Longrightarrow$
$\exists$volatile sop A L R W. Write$_{\mathsf{sb}}$ volatile a sop v A L R W $\in$ set sb
apply (induct sb)
apply simp
apply (clarsimp split: memref.splits option.splits if-split-asm)
apply blast
apply blast
done

lemma flush-append-Read$_{\mathsf{sb}}$:
$\bigwedge$m. (flush (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) (sb @ [Read$_{\mathsf{sb}}$ volatile a t v])) m)
= flush (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) m

by (induct sb) (auto split: memref.splits)

lemma flush-append-write:
$\bigwedge$m. (flush (sb @ [$\text{Write}_{\mathsf{sb}}$ volatile a sop v A L R W]) m) = (flush sb m) (a:=v)
by (induct sb) (auto split: memref.splits)

lemma flush-append-$\text{Prog}_{\mathsf{sb}}$:
$\bigwedge$m. (flush (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) (sb @ [$\text{Prog}_{\mathsf{sb}}$ p_1 p_2 mis])) m) =
(flush (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) m)
by (induct sb) (auto split: memref.splits)

lemma flush-append-$\text{Ghost}_{\mathsf{sb}}$:
$\bigwedge$m. (flush (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) (sb @ [$\text{Ghost}_{\mathsf{sb}}$ A L R W])) m) =
(flush (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) m)
by (induct sb) (auto split: memref.splits)

lemma share-append: $\bigwedge$S. share (xs@ys) S = share ys (share xs S)
by (induct xs) (auto split: memref.splits)

lemma share-append-$\text{Read}_{\mathsf{sb}}$:
$\bigwedge$S. (share (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) (sb @ [$\text{Read}_{\mathsf{sb}}$ volatile a t v])) S)
= share (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) S
by (induct sb) (auto split: memref.splits)

lemma share-append-$\text{Write}_{\mathsf{sb}}$:
$\bigwedge$S. (share (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) (sb @ [$\text{Write}_{\mathsf{sb}}$ volatile a sop v A L R W])) S)
= share (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) S
by (induct sb) (auto split: memref.splits)

lemma share-append-$\text{Prog}_{\mathsf{sb}}$:
$\bigwedge$S. (share (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) (sb @ [$\text{Prog}_{\mathsf{sb}}$ p_1 p_2 mis])) S) =
(share (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) S)
by (induct sb) (auto split: memref.splits)

lemma in-acquired-no-pending-write-outstanding-write:
a ∈ acquired False sb A $\Longrightarrow$ outstanding-refs is-volatile-$\text{Write}_{\mathsf{sb}}$ sb ≠ {}
apply (induct sb)
apply (auto split: memref.splits)
done

lemma flush-buffered-val-conv:
$\bigwedge$m. flush sb m a = (case buffered-val sb a of None ⇒ m a | Some v ⇒ v)
by (induct sb) (auto split: memref.splits option.splits)

lemma reads-consistent-unbuffered-snoc:
$\bigwedge$m. buffered-val sb a = None $\Longrightarrow$ m a = v $\Longrightarrow$ reads-consistent pending-write $\mathcal{O}$ m sb
$\Longrightarrow$

volatile ⟶
outstanding-refs is-volatile-Write$_{sb}$ sb = {}
⟹ reads-consistent pending-write $\mathcal{O}$ m (sb @ [Read$_{sb}$ volatile a t v])
by (simp add: reads-consistent-append flush-buffered-val-conv)

lemma reads-consistent-buffered-snoc:
⋀m. buffered-val sb a = Some v ⟹ reads-consistent pending-write $\mathcal{O}$ m sb ⟹
volatile ⟶ outstanding-refs is-volatile-Write$_{sb}$ sb = {}
⟹ reads-consistent pending-write $\mathcal{O}$ m (sb @ [Read$_{sb}$ volatile a t v])
by (simp add: reads-consistent-append flush-buffered-val-conv)

lemma reads-consistent-snoc-Write$_{sb}$:
⋀m. reads-consistent pending-write $\mathcal{O}$ m sb ⟹
reads-consistent pending-write $\mathcal{O}$ m (sb @ [Write$_{sb}$ volatile a sop v A L R W])
by (simp add: reads-consistent-append)

lemma reads-consistent-snoc-Prog$_{sb}$:
⋀m. reads-consistent pending-write $\mathcal{O}$ m sb ⟹ reads-consistent pending-write $\mathcal{O}$ m (sb @ [Prog$_{sb}$ p_1 p_2 mis])
by (simp add: reads-consistent-append)

lemma reads-consistent-snoc-Ghost$_{sb}$:
⋀m. reads-consistent pending-write $\mathcal{O}$ m sb ⟹ reads-consistent pending-write $\mathcal{O}$ m (sb @ [Ghost$_{sb}$ A L R W])
by (simp add: reads-consistent-append)

lemma restrict-map-id [simp]:m |‘ dom m = m
apply (rule ext)
subgoal for x
apply (case-tac m x)
apply (auto simp add: restrict-map-def domIff)
done
done

lemma flush-all-until-volatile-write-Read-commute:
shows ⋀m i. ⟦i < length ls; ls!i=(p,Read volatile a t#is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
⟧
⟹
flush-all-until-volatile-write
(ls[i := (p,is , j(t↦v), sb @ [Read$_{sb}$ volatile a t v],$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)]) m =
flush-all-until-volatile-write ls m
proof (induct ls)
case Nil **thus** ?case
by simp
next
case (Cons l ls)
note i-bound = ‹i < length (l#ls)›
note ith = ‹(l#ls)!i = (p,Read volatile a t#is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)›
show ?case

```
  proof (cases i)
    case 0
    from ith 0 have l: l = (p,Read volatile a t#is,j,sb,𝒟,𝒪,ℛ)
      by simp
    thus ?thesis
      by (simp add: 0 flush-append-Read_sb del: fun-upd-apply )
  next
    case (Suc n)
    obtain p_l is_l 𝒪_l ℛ_l 𝒟_l j_l sb_l where l: l = (p_l,is_l,j_l,sb_l,𝒟_l,𝒪_l,ℛ_l)
      by (cases l)
    from i-bound ith
    have flush-all-until-volatile-write
      (ls[n := (p,is , j(t↦v), sb @ [Read_sb volatile a t v],𝒟',𝒪',ℛ') ])
      (flush (takeWhile (Not ∘ is-volatile-Write_sb) sb_l) m) =
      flush-all-until-volatile-write ls (flush (takeWhile (Not ∘ is-volatile-Write_sb) sb_l) m)
      apply −
      apply (rule Cons.hyps)
      apply (auto simp add: Suc l)
      done

    then
    show ?thesis
      by (simp add: Suc l del: fun-upd-apply)
  qed
qed

lemma flush-all-until-volatile-write-append-Ghost-commute:
  ⋀i m. ⟦i < length ts; ts!i=(p,is,j,sb,𝒟,𝒪,ℛ)⟧
    ⟹ flush-all-until-volatile-write (ts[i := (p',is',j', sb@[Ghost_sb A L R W], 𝒟', 𝒪',ℛ')])
m
      = flush-all-until-volatile-write ts m
proof (induct ts)
  case Nil thus ?case
    by simp
next
  case (Cons l ts)
  note i-bound =  ‹i < length (l#ts)›
  note ith = ‹(l#ts)!i = (p,is,j,sb,𝒟,𝒪,ℛ)›
  show ?case
  proof (cases i)
    case 0
    from ith 0 have l: l = (p,is,j,sb,𝒟,𝒪,ℛ)
      by simp
    thus ?thesis
      by (simp add: 0 flush-append-Ghost_sb del: fun-upd-apply)
  next
    case (Suc n)
    obtain p_l is_l 𝒪_l ℛ_l 𝒟_l j_l sb_l where l: l = (p_l,is_l,j_l,sb_l,𝒟_l,𝒪_l,ℛ_l)
      by (cases l)
```

from i-bound ith
have flush-all-until-volatile-write
(ts[n := (p′,is′,j′, sb@[Ghost$_{sb}$ A L R W], $\mathcal{D}'$, $\mathcal{O}'$,$\mathcal{R}'$)])
(flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_l$) m) =
flush-all-until-volatile-write ts
(flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_l$) m)
apply −
apply (rule Cons.hyps)
apply (auto simp add: Suc l)
done

then show ?thesis
by (simp add: Suc l)
qed
qed

lemma update-commute:
assumes g-unchanged: ∀a m. a ∉ G ⟶ g m a = m a
assumes g-independent: ∀a m. a ∈ G ⟶ g (f m) a = g m a
assumes f-unchanged: ∀a m. a ∉ F ⟶ f m a = m a
assumes f-independent: ∀a m. a ∈ F ⟶ f (g m) a = f m a
assumes disj: G ∩ F = {}
shows f (g m) = g (f m)
proof
fix a
show f (g m) a = g (f m) a
proof (cases a ∈ G)
case True
with disj **have** a-notin-F: a ∉ F
by blast
from f-unchanged [rule-format, OF a-notin-F, of g m]
have f (g m) a = g m a **.**
also
from g-independent [rule-format, OF True]
have . . . = g (f m) a **by** simp
finally show ?thesis **.**
next
case False
note a-notin-G = this
show ?thesis
proof (cases a ∈ F)
case True
from f-independent [rule-format, OF True]
have f (g m) a = f m a **by** simp
also
from g-unchanged [rule-format, OF a-notin-G]
have . . . = g (f m) a
by simp
finally show ?thesis **.**

next
case False
from f-unchanged [rule-format, OF False]
have f (g m) a = g m a.
also
from g-unchanged [rule-format, OF a-notin-G]
have ... = m a .
also
from f-unchanged [rule-format, OF False]
have ... = f m a **by** simp
also
from g-unchanged [rule-format, OF a-notin-G]
have ... = g (f m) a
by simp
finally show ?thesis .
qed
qed
qed

lemma update-commute′:
assumes g-unchanged: $\forall$a m. a $\notin$ G $\longrightarrow$ g m a = m a
assumes g-independent: $\forall$a m_1 m_2. a $\in$ G $\longrightarrow$ g m_1 a = g m_2 a
assumes f-unchanged: $\forall$a m. a $\notin$ F $\longrightarrow$ f m a = m a
assumes f-independent: $\forall$a m_1 m_2. a $\in$ F $\longrightarrow$ f m_1 a = f m_2 a
assumes disj: G $\cap$ F = {}
shows f (g m) = g (f m)
proof −
from g-independent **have** g-ind′: $\forall$a m. a $\in$ G $\longrightarrow$ g (f m) a = g m a **by** blast
from f-independent **have** f-ind′: $\forall$a m. a $\in$ F $\longrightarrow$ f (g m) a = f m a **by** blast
from update-commute [OF g-unchanged g-ind′ f-unchanged f-ind′ disj]
show ?thesis .
qed

lemma flush-unchanged-addresses: $\bigwedge$m. a $\notin$ outstanding-refs is-Write$_{sb}$ sb $\Longrightarrow$ flush sb m a = m a
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons r sb)
note a-notin = ‹a $\notin$ outstanding-refs is-Write$_{sb}$ (r#sb)›
show ?case
proof (cases r)
case (Write$_{sb}$ volatile a′ sop v)
from a-notin **obtain** neq-a-a′: a≠a′ **and** a-notin′: a $\notin$ outstanding-refs is-Write$_{sb}$ sb **by** (simp add: Write$_{sb}$)
from Cons.hyps [OF a-notin′, of m(a′:=v)] neq-a-a′
show ?thesis
apply (simp add: Write$_{sb}$ del: fun-upd-apply)
apply simp

    done
  next
    case (Read$_{sb}$ volatile a′ t v)
    from a-notin obtain a-notin′: a ∉ outstanding-refs is-Write$_{sb}$ sb
      by (simp add: Read$_{sb}$)
    from Cons.hyps [OF a-notin′, of m]
    show ?thesis
      by (simp add: Read$_{sb}$)
  next
    case Prog$_{sb}$ with Cons show ?thesis by simp
  next
    case Ghost$_{sb}$ with Cons show ?thesis by simp
  qed
qed

lemma flushed-values-mem-independent:
  ⋀m m′ a. a ∈ outstanding-refs is-Write$_{sb}$ sb ⟹ flush sb m′ a = flush sb m a
proof (induct sb)
  case Nil thus ?case by simp
next
  case (Cons r sb)
  show ?case
  proof (cases r)
    case (Write$_{sb}$ volatile a′ sop′ v′)
    have flush sb (m′(a′ := v′)) a′ = flush sb (m(a′ := v′)) a′
    proof (cases a′ ∈ outstanding-refs is-Write$_{sb}$ sb)
      case True
      from Cons.hyps [OF this]
      show ?thesis .
    next
      case False
      from flush-unchanged-addresses [OF False]
      show ?thesis
 by simp
    qed
    with Cons.hyps Cons.prems
    show ?thesis
      by (auto simp add: Write$_{sb}$)
  next
    case Read$_{sb}$ thus ?thesis using Cons
      by auto
  next
    case Prog$_{sb}$ thus ?thesis using Cons
      by auto
  next
    case Ghost$_{sb}$ thus ?thesis using Cons
      by auto
  qed
qed

lemma flush-all-until-volatile-write-unchanged-addresses:
  $\bigwedge$m. a $\notin$ $\bigcup$((λ(-,-,-,sb,-,-,-). outstanding-refs is-Write$_{sb}$
    (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)) ' set ls) $\Longrightarrow$
    flush-all-until-volatile-write ls m a = m a
proof (induct ls)
  case Nil **thus** ?case **by** simp
next
  case (Cons l ls)
  obtain p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ xs sb **where** l: l=(p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
    by (cases l)
  note ‹a $\notin$ $\bigcup$ ((λ(-,-,-,sb,-,-,-). outstanding-refs is-Write$_{sb}$
    (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)) ' set (l#ls))›
  then obtain
    a-notin-sb: a $\notin$ outstanding-refs is-Write$_{sb}$
      (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) **and**
    a-notin-ls: a $\notin$ $\bigcup$ ((λ(-,-,-,sb,-,-,-). outstanding-refs is-Write$_{sb}$
      (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)) ' set ls)
    by (auto simp add: l)

  from Cons.hyps [OF a-notin-ls]
  have flush-all-until-volatile-write ls (flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) m)
a
    =
    (flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) m) a**.**

  also

  from flush-unchanged-addresses [OF a-notin-sb]
  have (flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) m) a = m a**.**
  finally
  show ?case
    by (simp add: l)
qed

lemma notin-outstanding-non-volatile-takeWhile-lem:
  a $\notin$ outstanding-refs (Not ∘ is-volatile) sb
    $\Longrightarrow$
    a $\notin$ outstanding-refs is-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
apply (induct sb)
apply (auto simp add: is-Write$_{sb}$-def split: if-split-asm memref.splits)
done

lemma notin-outstanding-non-volatile-takeWhile-lem′:
  a $\notin$ outstanding-refs is-non-volatile-Write$_{sb}$ sb
    $\Longrightarrow$
    a $\notin$ outstanding-refs is-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
apply (induct sb)
apply (auto simp add: is-Write$_{sb}$-def split: if-split-asm memref.splits)
done

lemma notin-outstanding-non-volatile-takeWhile-Un-lem$'$:
a $\notin \bigcup$ ((λ(-,-,-,sb,-,-,-). outstanding-refs (Not $\circ$ is-volatile) sb) ' set ls)
 $\Longrightarrow$ a $\notin \bigcup$ ((λ(-,-,-,sb,-,-,-). outstanding-refs is-Write$_{\mathsf{sb}}$
 (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb)) ' set ls)
proof (induct ls)
 case Nil **thus** ?case **by** simp
next
 case (Cons l ls)
 obtain p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ xs sb **where** l: l=(p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  by (cases l)

 from Cons.prems
 obtain
  a-notin-sb: a $\notin$ outstanding-refs (Not $\circ$ is-volatile) sb **and**
  a-notin-ls: a $\notin \bigcup$ ((λ(-,-,-,sb,-,-,-). outstanding-refs (Not $\circ$ is-volatile) sb) ' set ls)
  by (force simp add: l simp del: o-apply)
 from notin-outstanding-non-volatile-takeWhile-lem [OF a-notin-sb]
  Cons.hyps [OF a-notin-ls]
 show ?case
  by (auto simp add: l simp del: o-apply)
qed

lemma flush-all-until-volatile-write-unchanged-addresses$'$:
 assumes notin: a $\notin \bigcup$ ((λ(-,-,-,sb,-,-,-). outstanding-refs (Not $\circ$ is-volatile) sb) ' set ls)
 shows flush-all-until-volatile-write ls m a = m a
using notin-outstanding-non-volatile-takeWhile-Un-lem$'$ [OF notin]
by (auto intro: flush-all-until-volatile-write-unchanged-addresses)

lemma flush-all-until-volatile-wirte-mem-independent:
 $\bigwedge$m m$'$. a $\in \bigcup$ ((λ(-,-,-,sb,-,-,-). outstanding-refs is-Write$_{\mathsf{sb}}$
 (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb)) ' set ls) $\Longrightarrow$
 flush-all-until-volatile-write ls m$'$ a = flush-all-until-volatile-write ls m a
proof (induct ls)
 case Nil **thus** ?case **by** simp
next
 case (Cons l ls)
 obtain p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ xs sb **where** l: l=(p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  by (cases l)
 note a-in = ‹a $\in \bigcup$ ((λ(-,-,-,sb,-,-,-). outstanding-refs is-Write$_{\mathsf{sb}}$
 (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb)) ' set (l#ls))›
 show ?case
 proof (cases a $\in \bigcup$ ((λ(-,-,-,sb,-,-,-). outstanding-refs is-Write$_{\mathsf{sb}}$
 (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb)) ' set ls))
  case True
  from Cons.hyps [OF this]
  show ?thesis
   by (simp add: l)
 next
  case False

with a-in
have a ∈ outstanding-refs is-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
by (auto simp add: l)
from flushed-values-mem-independent [rule-format, OF this]
have flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) m′ a =
flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) m a.
with flush-all-until-volatile-write-unchanged-addresses [OF False]
show ?thesis
by (auto simp add: l)
qed
qed

lemma flush-all-until-volatile-write-buffered-val-conv:
assumes no-volatile-Write$_{sb}$: outstanding-refs is-volatile-Write$_{sb}$ sb = {}
shows⋀m i. ⟦i < length ls; ls!i = (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);

∀ j < length ls. i ≠ j ⟶
(let (-,-,-,sb$_j$,-,-,-) = ls!j
in a ∉ outstanding-refs is-non-volatile-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$)) ⟧ ⟹
flush-all-until-volatile-write ls m a =
(case buffered-val sb a of None ⇒ m a | Some v ⇒ v)
proof (induct ls)
case Nil **thus** ?case
by simp
next
case (Cons l ls)
note i-bound = ‹i < length (l#ls)›
note ith = ‹(l#ls)!i = (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)›
note notin = ‹∀ j < length (l#ls). i ≠ j ⟶
(let (-,-,-,sb$_j$,-,-,-) = (l#ls)!j
in a ∉ outstanding-refs is-non-volatile-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$))›
show ?case
proof (cases i)
case 0
from ith 0 **have** l: l = (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by simp
from no-volatile-Write$_{sb}$ **have** take-all: takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb = sb
by (auto simp add: outstanding-refs-conv)

have a ∉ ⋃((λ(-,-, -, sb, -,-,-).
outstanding-refs is-Write$_{sb}$
(takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)) ‘ set ls) (**is** a ∉ ?LS)
proof
assume a ∈ ?LS
from in-Union-image-nth-conv [OF this]
obtain j p$_j$ is$_j$ $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs$_j$ sb$_j$ **where**
j-bound: j < length ls **and**
jth: ls!j = (p$_j$,is$_j$,xs$_j$,sb$_j$,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$) **and**

a-in-j: a ∈ outstanding-refs is-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$)
by fastforce
from a-in-j **obtain** v′ sop′ A L R W **where** Write$_{sb}$ False a sop′ v′ A L R W∈ set (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$)
apply (clarsimp simp add: outstanding-refs-conv)
subgoal for x
apply (case-tac x)
apply clarsimp
apply (frule set-takeWhileD)
apply auto
done
done
with notin [rule-format, of Suc j] j-bound jth
show False
by (force simp add: 0 outstanding-refs-conv is-non-volatile-Write$_{sb}$-def
split: memref.splits)
qed
from flush-all-until-volatile-write-unchanged-addresses [OF this]
have flush-all-until-volatile-write ls (flush sb m) a = (flush sb m) a
by (simp add: take-all)
then
show ?thesis
by (simp add: 0 l take-all flush-buffered-val-conv)
next
case (Suc n)
obtain p$_l$ is$_l$ $\mathcal{O}_l$ $\mathcal{R}_l$ $\mathcal{D}_l$ xs$_l$ sb$_l$ **where** l: l = (p$_l$,is$_l$,xs$_l$,sb$_l$,$\mathcal{D}_l$,$\mathcal{O}_l$,$\mathcal{R}_l$)
by (cases l)

from i-bound ith notin
have flush-all-until-volatile-write ls
(flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_l$) m) a
= (case buffered-val sb a of None ⇒
(flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_l$) m) a | Some v ⇒ v)
apply −
apply (rule Cons.hyps)
apply (force simp add: Suc Let-def simp del: o-apply)+
done

moreover
from notin [rule-format, of 0] l
have a ∉ outstanding-refs is-non-volatile-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_l$)
by (auto simp add: Let-def outstanding-refs-conv Suc)
then
have a ∉ outstanding-refs is-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_l$)
apply (clarsimp simp add: outstanding-refs-conv is-Write$_{sb}$-def split: memref.splits dest: set-takeWhileD)
apply (frule set-takeWhileD)
apply force
done

from flush-unchanged-addresses [OF this]
have (flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_l$) m) a = m a **.**

ultimately
show ?thesis
by (simp add: Suc l split: option.splits)
qed
qed

context program
begin

abbreviation sb-concurrent-step ::
('p,'p store-buffer,'dirty,'owns,'rels,'shared) global-config ⇒ ('p,'p store-buffer,'dirty,'owns,'rels,'shared) global-config ⇒ bool
(‹- ⇒$_{sb}$ -› [60,60] 100)
where
sb-concurrent-step ≡
computation.concurrent-step sb-memop-step store-buffer-step program-step (λp p' is sb. sb)

term x ⇒$_{sb}$ Y

abbreviation (**in** program) sb-concurrent-steps::
('p,'p store-buffer,'dirty,'owns,'rels,'shared) global-config ⇒ ('p,'p store-buffer,'dirty,'owns,'rels,'shared) global-config ⇒ bool
(‹- ⇒$_{sb}$* -› [60,60] 100)
where
sb-concurrent-steps ≡ sb-concurrent-step^**

term x ⇒$_{sb}$* Y

abbreviation sbh-concurrent-step ::
('p,'p store-buffer,bool,owns,rels,shared) global-config ⇒ ('p,'p store-buffer,bool,owns,rels,shared) global-config ⇒ bool
(‹- ⇒$_{sbh}$ -› [60,60] 100)
where
sbh-concurrent-step ≡
computation.concurrent-step sbh-memop-step flush-step program-step
(λp p' is sb. sb @ [Prog$_{sb}$ p p' is])

term x ⇒$_{sbh}$ Y

abbreviation sbh-concurrent-steps::
('p,'p store-buffer,bool,owns,rels,shared) global-config ⇒ ('p,'p store-buffer,bool,owns,rels,shared) global-config ⇒ bool
(‹- ⇒$_{sbh}$* -› [60,60] 100)
where

sbh-concurrent-steps ≡ sbh-concurrent-step^**

term x $\Rightarrow_{\mathsf{sbh}}{}^*$ Y
end

lemma instrs-append-Read$_{\mathsf{sb}}$:
instrs (sb@[Read$_{\mathsf{sb}}$ volatile a t v]) = instrs sb @ [Read volatile a t]
by (induct sb) (auto split: memref.splits)

lemma instrs-append-Write$_{\mathsf{sb}}$:
instrs (sb@[Write$_{\mathsf{sb}}$ volatile a sop v A L R W]) = instrs sb @ [Write volatile a sop A L R W]
by (induct sb) (auto split: memref.splits)

lemma instrs-append-Ghost$_{\mathsf{sb}}$:
instrs (sb@[Ghost$_{\mathsf{sb}}$ A L R W]) = instrs sb @ [Ghost A L R W]
by (induct sb) (auto split: memref.splits)

lemma prog-instrs-append-Ghost$_{\mathsf{sb}}$:
prog-instrs (sb@[Ghost$_{\mathsf{sb}}$ A L R W]) = prog-instrs sb
by (induct sb) (auto split: memref.splits)

lemma prog-instrs-append-Read$_{\mathsf{sb}}$:
prog-instrs (sb@[Read$_{\mathsf{sb}}$ volatile a t v]) = prog-instrs sb
by (induct sb) (auto split: memref.splits)

lemma prog-instrs-append-Write$_{\mathsf{sb}}$:
prog-instrs (sb@[Write$_{\mathsf{sb}}$ volatile a sop v A L R W]) = prog-instrs sb
by (induct sb) (auto split: memref.splits)

lemma hd-prog-append-Read$_{\mathsf{sb}}$:
hd-prog p (sb@[Read$_{\mathsf{sb}}$ volatile a t v]) = hd-prog p sb
by (induct sb) (auto split: memref.splits)

lemma hd-prog-append-Write$_{\mathsf{sb}}$:
hd-prog p (sb@[Write$_{\mathsf{sb}}$ volatile a sop v A L R W]) = hd-prog p sb
by (induct sb) (auto split: memref.splits)

lemma flush-update-other: $\bigwedge$m. a ∉ outstanding-refs (Not ∘ is-volatile) sb $\Longrightarrow$
outstanding-refs (is-volatile-Write$_{\mathsf{sb}}$) sb = {} $\Longrightarrow$
flush sb (m(a:=v)) = (flush sb m)(a := v)
by (induct sb)
(auto split: memref.splits if-split-asm simp add: fun-upd-twist)

lemma flush-update-other′: $\bigwedge$m. a ∉ outstanding-refs (is-non-volatile-Write$_{\mathsf{sb}}$) sb $\Longrightarrow$
outstanding-refs (is-volatile-Write$_{\mathsf{sb}}$) sb = {} $\Longrightarrow$
flush sb (m(a:=v)) = (flush sb m)(a := v)
by (induct sb)
(auto split: memref.splits if-split-asm simp add: fun-upd-twist)

lemma flush-update-other′′: $\bigwedge$m. a $\notin$ outstanding-refs (is-non-volatile-Write$_{\mathsf{sb}}$) sb $\Longrightarrow$
a $\notin$ outstanding-refs (is-volatile-Write$_{\mathsf{sb}}$) sb $\Longrightarrow$
flush sb (m(a:=v)) = (flush sb m)(a := v)
by (induct sb)
(auto split: memref.splits if-split-asm simp add: fun-upd-twist)

lemma flush-all-until-volatile-write-update-other:
$\bigwedge$m. $\forall$ j < length ts.
(let (-,-,-,sb$_j$,-,-,-) = ts!j
in a $\notin$ outstanding-refs is-non-volatile-Write$_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_j$))
$\Longrightarrow$
flush-all-until-volatile-write ts (m(a := v)) =
(flush-all-until-volatile-write ts m)(a := v)
proof (induct ts)
case Nil **thus** ?case
by simp
next
case (Cons t ts)
note notin = ‹$\forall$ j < length (t#ts).
(let (-,-,-,sb$_j$,-,-,-) = (t#ts)!j
in a $\notin$ outstanding-refs is-non-volatile-Write$_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_j$))›
hence notin′: $\forall$ j < length ts.
(let (-,-,-,sb$_j$,-,-,-) = ts!j
in a $\notin$ outstanding-refs is-non-volatile-Write$_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_j$))
by auto

obtain p$_l$ is$_l$ $\mathcal{O}_l$ $\mathcal{R}_l$ $\mathcal{D}_l$ xs$_l$ sb$_l$ **where** t: t = (p$_l$,is$_l$,xs$_l$,sb$_l$,$\mathcal{D}_l$,$\mathcal{O}_l$,$\mathcal{R}_l$)
by (cases t)

have no-write:
outstanding-refs (is-volatile-Write$_{\mathsf{sb}}$) (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_l$) = {}
by (auto simp add: outstanding-refs-conv dest: set-takeWhileD)

from notin [rule-format, of 0] t
have a-notin:
a $\notin$ outstanding-refs is-non-volatile-Write$_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_l$)
by (auto)

from flush-update-other′ [OF a-notin no-write]
have (flush (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_l$) (m(a := v))) =
(flush (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_l$) m)(a := v).
with Cons.hyps [OF notin′, of (flush (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_l$) m)]
show ?case
by (simp add: t del: fun-upd-apply)
qed

lemma flush-all-until-volatile-write-append-non-volatile-write-commute:
assumes no-volatile-Write$_{\mathsf{sb}}$: outstanding-refs is-volatile-Write$_{\mathsf{sb}}$ sb = {}
shows $\bigwedge$m i. ⟦i < length ts; ts!i = (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
$\forall$j < length ts. i $\neq$ j $\longrightarrow$
(let (-,-,-,sb$_j$,-,-,-) = ts!j
in a $\notin$ outstanding-refs is-non-volatile-Write$_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_j$))⟧
$\Longrightarrow$ flush-all-until-volatile-write (ts[i := (p′,is′, xs, sb @ [Write$_{\mathsf{sb}}$ False a sop v A L R W],$\mathcal{D}'$, $\mathcal{O}$,$\mathcal{R}'$)]) m =
(flush-all-until-volatile-write ts m)(a := v)
proof (induct ts)
case Nil **thus** ?case
by simp
next
case (Cons t ts)
note i-bound = ‹i < length (t#ts)›
note ith = ‹(t#ts)!i = (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)›
note notin = ‹$\forall$j < length (t#ts). i $\neq$ j $\longrightarrow$
(let (-,-,-,sb$_j$,-,-,-) = (t#ts)!j
in a $\notin$ outstanding-refs is-non-volatile-Write$_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_j$))›
show ?case
proof (cases i)
case 0
from ith 0 **have** t: t = (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by simp
from no-volatile-Write$_{\mathsf{sb}}$ **have** take-all: takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb = sb
by (auto simp add: outstanding-refs-conv)

from no-volatile-Write$_{\mathsf{sb}}$
have take-all′: takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) (sb @ [Write$_{\mathsf{sb}}$ False a sop v A L R W]) =
(sb @ [Write$_{\mathsf{sb}}$ False a sop v A L R W])
by (auto simp add: outstanding-refs-conv)
from notin
have $\forall$j < length ts.
(let (-,-,-,sb$_j$,-,-,-) = ts!j
in a $\notin$ outstanding-refs is-non-volatile-Write$_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_j$))
by (auto simp add: 0)

from flush-all-until-volatile-write-update-other [OF this]
show ?thesis
by (simp add: 0 t take-all′ take-all flush-append-write del: fun-upd-apply)
next
case (Suc n)
obtain p$_l$ is$_l$ $\mathcal{O}_l$ $\mathcal{R}_l$ $\mathcal{D}_l$ xs$_l$ sb$_l$ **where** t: t = (p$_l$,is$_l$,xs$_l$,sb$_l$,$\mathcal{D}_l$,$\mathcal{O}_l$,$\mathcal{R}_l$)
by (cases t)
from i-bound ith notin
have flush-all-until-volatile-write

(ts[n := (p',is',xs, sb @ [Write$_{sb}$ False a sop v A L R W], $\mathcal{D}'$, $\mathcal{O}$,$\mathcal{R}'$)])
(flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_l$) m) =
(flush-all-until-volatile-write ts
(flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_l$) m))
(a := v)
apply –
apply (rule Cons.hyps)
apply (auto simp add: Suc simp del: o-apply)
done

then
show ?thesis
by (simp add: t Suc del: fun-upd-apply)
qed
qed

lemma flush-all-until-volatile-write-append-unflushed:
assumes volatile-Write$_{sb}$: ¬ outstanding-refs is-volatile-Write$_{sb}$ sb = {}
shows $\bigwedge$m i. ⟦i < length ts; ts!i = (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)⟧
⟹ flush-all-until-volatile-write (ts[i := (p',is', xs, sb @ sbx,$\mathcal{D}'$, $\mathcal{O}$,$\mathcal{R}'$)]) m =
(flush-all-until-volatile-write ts m)
proof (induct ts)
case Nil **thus** ?case
by simp
next
case (Cons l ts)
note i-bound = ‹i < length (l#ts)›
note ith = ‹(l#ts)!i = (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)›
show ?case
proof (cases i)
case 0
from ith 0 **have** l: l = (p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by simp
from volatile-Write$_{sb}$
obtain r **where** r-in: r ∈ set sb **and** volatile-r: is-volatile-Write$_{sb}$ r
by (auto simp add: outstanding-refs-conv)
from takeWhile-append1 [OF r-in, of (Not ∘ is-volatile-Write$_{sb}$)] volatile-r

have (flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) (sb @ sbx)) m) =
(flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) m)
by auto
then
show ?thesis
by (simp add: 0 l)
next
case (Suc n)
obtain p$_l$ is$_l$ $\mathcal{O}_l$ $\mathcal{R}_l$ $\mathcal{D}_l$ xs$_l$ sb$_l$ **where** l: l = (p$_l$,is$_l$,xs$_l$,sb$_l$,$\mathcal{D}_l$,$\mathcal{O}_l$,$\mathcal{R}_l$)
by (cases l)

from Cons.hyps [of n] i-bound ith

show ?thesis
by (simp add: l Suc)
qed
qed

lemma flush-all-until-volatile-nth-update-unused:
shows $\bigwedge$m i. $[\![$i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)$]\!]$
$\Longrightarrow$ flush-all-until-volatile-write (ts[i := (p′,is′,j′, sb, $\mathcal{D}'$, $\mathcal{O}'$,$\mathcal{R}'$)]) m =
(flush-all-until-volatile-write ts m)
proof (induct ts)
case Nil **thus** ?case
by simp
next
case (Cons l ts)
note i-bound = ‹i < length (l#ts)›
note ith = ‹(l#ts)!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)›
show ?case
proof (cases i)
case 0
from ith 0 **have** l: l = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by simp
show ?thesis
by (simp add: 0 l)
next
case (Suc n)
obtain p_l is_l $\mathcal{O}_l$ $\mathcal{R}_l$ $\mathcal{D}_l$ j_l sb_l **where** l: l = (p_l,is_l,j_l,sb_l,$\mathcal{D}_l$,$\mathcal{O}_l$,$\mathcal{R}_l$)
by (cases l)

from Cons.hyps [of n] i-bound ith
show ?thesis
by (simp add: l Suc)
qed
qed

lemma flush-all-until-volatile-write-append-volatile-write-commute:
assumes no-volatile-Write$_{sb}$: outstanding-refs is-volatile-Write$_{sb}$ sb = {}
shows $\bigwedge$m i. $[\![$i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)$]\!]$ $\Longrightarrow$
flush-all-until-volatile-write
(ts[i := (p′,is′, j, sb @ [Write$_{sb}$ True a sop v A L R W],$\mathcal{D}'$, $\mathcal{O}$,$\mathcal{R}'$)]) m
= flush-all-until-volatile-write ts m
proof (induct ts)
case Nil **thus** ?case
by simp
next
case (Cons l ts)
note i-bound = ‹i < length (l#ts)›
note ith = ‹(l#ts)!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)›
show ?case
proof (cases i)
case 0

from ith 0 **have** l: l = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  by simp
from no-volatile-Write$_{\mathsf{sb}}$
have s1: takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb = sb
  by (auto simp add: outstanding-refs-conv)

from no-volatile-Write$_{\mathsf{sb}}$
have s2: (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) (sb @ [Write$_{\mathsf{sb}}$ True a sop v A L R W]))
= sb
  by (auto simp add: outstanding-refs-conv)

show ?thesis
  by (simp add: 0 l s1 s2)
next
  case (Suc n)
  obtain p_l is_l $\mathcal{O}_l$ $\mathcal{R}_l$ $\mathcal{D}_l$ j_l sb_l **where** l: l = (p_l,is_l,j_l,sb_l,$\mathcal{D}_l$,$\mathcal{O}_l$,$\mathcal{R}_l$)
    by (cases l)

  from Cons.hyps [of n] i-bound ith
  show ?thesis
    by (simp add: l Suc)
qed
qed

lemma reads-consistent-update:
$\bigwedge$pending-write $\mathcal{O}$ m. reads-consistent pending-write $\mathcal{O}$ m sb $\Longrightarrow$
  a $\notin$ outstanding-refs (Not $\circ$ is-volatile) sb $\Longrightarrow$
  reads-consistent pending-write $\mathcal{O}$ (m(a := v)) sb
apply (induct sb)
apply simp
apply (clarsimp split: memref.splits if-split-asm
  simp add: fun-upd-twist)
subgoal for sb $\mathcal{O}$ m x11 addr val A R pending-write
apply (case-tac a=addr)
apply simp
apply (fastforce simp add: fun-upd-twist)
done
done

lemma (**in** program) history-consistent-hd-prog: $\bigwedge$p. history-consistent j p$'$ xs
$\Longrightarrow$ history-consistent j (hd-prog p xs) xs
apply (induct xs)
apply simp
apply (auto split: memref.splits option.splits)
done

locale valid-program = program +
fixes valid-prog
assumes valid-prog-inv: $[\![$j$\vdash$p $\to_{\mathsf{p}}$ (p$'$,is$'$); valid-prog p$]\!]$ $\Longrightarrow$ valid-prog p$'$

```
lemma (in valid-program) history-consistent-appendD:
  ⋀j ys p. ∀sop ∈ write-sops xs. valid-sop sop ⟹
               read-tmps xs ∩ read-tmps ys = {} ⟹
          history-consistent j p (xs@ys) ⟹
           (history-consistent (j|` (dom j − read-tmps ys)) p xs ∧
            history-consistent j (last-prog p xs) ys ∧
            read-tmps ys ∩ ⋃(fst ` write-sops xs) = {})
proof (induct xs)
  case Nil thus ?case
    by auto
next
  case (Cons x xs)
  note valid-sops = ‹∀sop∈write-sops (x # xs). valid-sop sop›
  note read-tmps-dist = ‹read-tmps (x#xs) ∩ read-tmps ys = {}›
  note consis = ‹history-consistent j p ((x#xs)@ys)›
  show ?case
  proof (cases x)
    case (Write_sb volatile a sop v)
    obtain D f where sop: sop=(D,f)
      by (cases sop)
    from consis obtain
      D-tmps: D ⊆ dom j and
      f-v: f j = v and
      D-read-tmps: D ∩ read-tmps (xs @ ys) = {} and
      consis′: history-consistent j p (xs @ ys)
      by (simp add: Write_sb sop)
    from valid-sops obtain
      valid-Df: valid-sop (D,f) and
      valid-sops′: ∀sop∈write-sops xs. valid-sop sop
      by (auto simp add: Write_sb sop)
    from valid-Df
    interpret valid-sop (D,f) .
    from read-tmps-dist have read-tmps-dist′: read-tmps xs ∩ read-tmps ys = {}
      by (simp add: Write_sb)


    from D-read-tmps have D-ys: D ∩ read-tmps ys = {}
      by (auto simp add: read-tmps-append)
    with D-tmps have D-subset: D ⊆ dom j − read-tmps ys
      by auto
    moreover

    from valid-sop [OF refl D-tmps]
    have f j = f (j |` D).
    moreover
    let ?j′ = j |` (dom j − read-tmps ys)
    from D-subset
    have ?j′ |` D = j |` D
      apply −
      apply (rule ext)
```

by (auto simp add: restrict-map-def)
moreover
from D-subset
have D-tmps′: D ⊆ dom ?j′
by auto
ultimately
have f-v′: f ?j′ = v
using valid-sop [OF refl D-tmps′] f-v
by simp
from D-read-tmps
have D ∩ read-tmps xs = {}
by (auto simp add: read-tmps-append)
with Cons.hyps [OF valid-sops′ read-tmps-dist′ consis′] D-tmps D-subset f-v′ D-ys
show ?thesis
by (auto simp add: $\mathrm{Write}_{\mathsf{sb}}$ sop)
next
case ($\mathrm{Read}_{\mathsf{sb}}$ volatile a t v)
from consis **obtain**
tmps-t: j t = Some v **and**
consis′: history-consistent j p (xs @ ys)
by (simp add: $\mathrm{Read}_{\mathsf{sb}}$ split: option.splits)

from read-tmps-dist
obtain t-ys: t ∉ read-tmps ys **and** read-tmps-dist′: read-tmps xs ∩ read-tmps ys = {}
by (auto simp add: $\mathrm{Read}_{\mathsf{sb}}$)
from valid-sops **have** valid-sops′: ∀sop∈write-sops xs. valid-sop sop
by (auto simp add: $\mathrm{Read}_{\mathsf{sb}}$)
from t-ys tmps-t
have (j |‘ (dom j − read-tmps ys)) t = Some v
by (auto simp add: restrict-map-def domIff)
with Cons.hyps [OF valid-sops′ read-tmps-dist′ consis′]

show ?thesis
by (auto simp add: $\mathrm{Read}_{\mathsf{sb}}$)
next
case ($\mathrm{Prog}_{\mathsf{sb}}$ p_1 p_2 mis)
from consis **obtain** p_1-p: p_1 = p **and**
prog-step: j |‘ (dom j − read-tmps (xs @ ys)) ⊢ p_1 $\rightarrow_{\mathsf{p}}$ (p_2, mis) **and**
consis′: history-consistent j p_2 (xs @ ys)
by (auto simp add: $\mathrm{Prog}_{\mathsf{sb}}$)

let ?j′ = j |‘ (dom j − read-tmps ys)
have eq: ?j′ |‘ (dom ?j′ − read-tmps xs) = j |‘ (dom j − read-tmps (xs @ ys))
apply (rule ext)
apply (auto simp add: read-tmps-append restrict-map-def domIff split: if-split-asm)
done

from valid-sops **have** valid-sops′: ∀sop∈write-sops xs. valid-sop sop
by (auto simp add: $\mathrm{Prog}_{\mathsf{sb}}$)
from read-tmps-dist

obtain read-tmps-dist′: read-tmps xs ∩ read-tmps ys = {}
by (auto simp add: $\text{Prog}_{\mathsf{sb}}$)
from Cons.hyps [OF valid-sops′ read-tmps-dist′ consis′] p_1-p prog-step eq
show ?thesis
by (simp add: $\text{Prog}_{\mathsf{sb}}$)
next
case $\text{Ghost}_{\mathsf{sb}}$
with Cons **show** ?thesis
by auto
qed
qed

lemma (**in** valid-program) history-consistent-appendI:
⋀j ys p. ∀sop ∈ write-sops xs. valid-sop sop ⟹
history-consistent (j|‘ (dom j − read-tmps ys)) p xs ⟹
history-consistent j (last-prog p xs) ys ⟹
read-tmps ys ∩ ⋃(fst ‘ write-sops xs) = {} ⟹ valid-prog p ⟹
history-consistent j p (xs@ys)
proof (induct xs)
case Nil **thus** ?case **by** simp
next
case (Cons x xs)
note valid-sops = ‹∀sop∈write-sops (x # xs). valid-sop sop›
note consis-xs = ‹history-consistent (j |‘ (dom j − read-tmps ys)) p (x # xs)›
note consis-ys = ‹history-consistent j (last-prog p (x # xs)) ys›
note dist = ‹read-tmps ys ∩ ⋃(fst ‘ write-sops (x # xs)) = {}›
note valid-p = ‹valid-prog p›
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a sop v)
obtain D f **where** sop: sop=(D,f)
by (cases sop)
from consis-xs **obtain**
D-tmps: D ⊆ dom j − read-tmps ys **and**
f-v: f (j |‘ (dom j − read-tmps ys)) = v (**is** f ?j = v) **and**
D-read-tmps: D ∩ read-tmps xs = {} **and**
consis′: history-consistent (j |‘ (dom j − read-tmps ys)) p xs
by (simp add: $\text{Write}_{\mathsf{sb}}$ sop)

from D-tmps D-read-tmps
have D ∩ read-tmps (xs @ ys) = {}
by (auto simp add: read-tmps-append)
moreover
from D-tmps **have** D-tmps′: D ⊆ dom j
by auto
moreover
from valid-sops **obtain**
valid-Df: valid-sop (D,f) **and**
valid-sops′: ∀sop∈write-sops xs. valid-sop sop
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ sop)

from valid-Df
interpret valid-sop (D,f) **.**

from D-tmps
have tmps-eq: j |‘ ((dom j − read-tmps ys) ∩ D) = j |‘ D
  apply −
  apply (rule ext)
  apply (auto simp add: restrict-map-def)
  done
from D-tmps
have f ?j = f (?j |‘ D)
  apply −
  apply (rule valid-sop [OF refl])
  apply auto
  done
with valid-sop [OF refl D-tmps′] f-v D-tmps

have f j = v
  by (clarsimp simp add: tmps-eq)
moreover
from consis-ys **have** consis-ys′: history-consistent j (last-prog p xs) ys
  by (auto simp add: $\text{Write}_{\mathsf{sb}}$)

from dist **have** dist′: read-tmps ys ∩ ⋃(fst ‘ write-sops xs) = {}
  by (auto simp add: $\text{Write}_{\mathsf{sb}}$)

moreover note Cons.hyps [OF valid-sops′ consis′ consis-ys′ dist′ valid-p]

ultimately show ?thesis
  by (simp add: $\text{Write}_{\mathsf{sb}}$ sop)
next
case ($\text{Read}_{\mathsf{sb}}$ volatile a t v)
from consis-xs **obtain**
  t-v: (j |‘ (dom j − read-tmps ys)) t = Some v **and**
  consis-xs′: history-consistent (j |‘ (dom j − read-tmps ys)) p xs
  by (clarsimp simp add: $\text{Read}_{\mathsf{sb}}$ split: option.splits)
from t-v **have** j t = Some v
  by (auto simp add: restrict-map-def split: if-split-asm)
moreover
from valid-sops **obtain**
  valid-sops′: ∀sop∈write-sops xs. valid-sop sop
  by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
from consis-ys **have** consis-ys′: history-consistent j (last-prog p xs) ys
  by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
from dist **have** dist′: read-tmps ys ∩ ⋃(fst ‘ write-sops xs) = {}
  by (auto simp add: $\text{Read}_{\mathsf{sb}}$)

note Cons.hyps [OF valid-sops′ consis-xs′ consis-ys′ dist′ valid-p]
ultimately
show ?thesis

by (simp add: $\mathsf{Read_{sb}}$)
next
case ($\mathsf{Prog_{sb}}$ p_1 p_2 mis)
let ?j = j |' (dom j − read-tmps ys)
from consis-xs **obtain**
p_1-p: p_1 = p **and**
prog-step: ?j |' (dom ?j − read-tmps xs) ⊢ $p_1 \rightarrow_{\mathsf{p}}$ (p_2, mis) **and**
consis′: history-consistent ?j p_2 xs
by (auto simp add: $\mathsf{Prog_{sb}}$)

have eq: ?j |' (dom ?j − read-tmps xs) = j |' (dom j − read-tmps (xs @ ys))
apply (rule ext)
apply (auto simp add: read-tmps-append restrict-map-def domIff split: if-split-asm)
done

from prog-step eq
have j |' (dom j − read-tmps (xs @ ys)) ⊢ $p_1 \rightarrow_{\mathsf{p}}$ (p_2, mis) **by** simp
moreover
from valid-sops **obtain**
valid-sops′: ∀sop∈write-sops xs. valid-sop sop
by (auto simp add: $\mathsf{Prog_{sb}}$)
from consis-ys **have** consis-ys′: history-consistent j (last-prog p_2 xs) ys
by (auto simp add: $\mathsf{Prog_{sb}}$)
from dist **have** dist′: read-tmps ys ∩ ⋃(fst ' write-sops xs) = {}
by (auto simp add: $\mathsf{Prog_{sb}}$)

note Cons.hyps [OF valid-sops′ consis′ consis-ys′ dist′ valid-prog-inv [OF prog-step valid-p [simplified p_1-p [symmetric]]]]
ultimately
show ?thesis
by (simp add: $\mathsf{Prog_{sb}}$ p_1-p)
next
case $\mathsf{Ghost_{sb}}$
with Cons **show** ?thesis
by auto
qed
qed

lemma (**in** valid-program) history-consistent-append-conv:
⋀j ys p. ∀sop ∈ write-sops xs. valid-sop sop ⟹
read-tmps xs ∩ read-tmps ys = {} ⟹ valid-prog p ⟹
history-consistent j p (xs@ys) =
(history-consistent (j|' (dom j − read-tmps ys)) p xs ∧
history-consistent j (last-prog p xs) ys ∧
read-tmps ys ∩ ⋃(fst ' write-sops xs) = {})
apply rule
apply (rule history-consistent-appendD,assumption+)
apply (rule history-consistent-appendI)
apply auto

done

lemma instrs-takeWhile-dropWhile-conv:
instrs xs = instrs (takeWhile P xs) @ instrs (dropWhile P xs)
by (induct xs) (auto split: memref.splits)

lemma (**in** program) history-consistent-hd-prog-p:
$\bigwedge$p. history-consistent j p xs $\Longrightarrow$ p = hd-prog p xs
by (induct xs) (auto split: memref.splits option.splits)

lemma instrs-append: $\bigwedge$ys. instrs (xs@ys) = instrs xs @ instrs ys
by (induct xs) (auto split: memref.splits)

lemma prog-instrs-append: $\bigwedge$ys. prog-instrs (xs@ys) = prog-instrs xs @ prog-instrs ys
by (induct xs) (auto split: memref.splits)

lemma prog-instrs-empty: $\forall$ r $\in$ set xs. $\neg$ is-Prog$_{sb}$ r $\Longrightarrow$ prog-instrs xs = []
by (induct xs) (auto split: memref.splits)

lemma length-dropWhile [termination-simp]: length (dropWhile P xs) $\leq$ length xs
by (induct xs) auto

lemma prog-instrs-filter-is-Prog$_{sb}$: prog-instrs (filter (is-Prog$_{sb}$) xs) = prog-instrs xs
by (induct xs) (auto split: memref.splits)

lemma Cons-to-snoc: $\bigwedge$x. $\exists$ ys y. (x#xs) = (ys@[y])
proof (induct xs)
case Nil **thus** ?case **by** simp
next
case (Cons x1 xs)
from Cons [of x1] **obtain** ys y **where** x1#xs = ys @ [y]
by auto
then
show ?case
by simp
qed

lemma causal-program-history-Read:
assumes causal-Read: causal-program-history (Read volatile a t # is$_{sb}$) sb
shows causal-program-history is$_{sb}$ (sb @ [Read$_{sb}$ volatile a t v])
proof
fix sb$_1$ sb$_2$
assume sb: sb @ [Read$_{sb}$ volatile a t v] = sb$_1$ @ sb$_2$
from causal-Read
interpret causal-program-history Read volatile a t # is$_{sb}$ sb **.**
show $\exists$ is. instrs sb$_2$ @ is$_{sb}$ = is @ prog-instrs sb$_2$
proof (cases sb$_2$)

case Nil
thus ?thesis
by simp
next
case (Cons r sb′)
from Cons-to-snoc [of r sb′] Cons **obtain** ys y **where** sb_2-snoc: sb_2=ys@[y]
by auto
with sb **obtain** y: y = $\text{Read}_{\mathsf{sb}}$ volatile a t v **and** sb: sb = sb_1@ys
by simp

from causal-program-history [OF sb] **obtain** is **where**
instrs ys @ Read volatile a t # is_{sb} = is @ prog-instrs ys
by auto
then show ?thesis
by (simp add: sb_2-snoc y instrs-append prog-instrs-append)
qed
qed

lemma causal-program-history-Write:
assumes causal-Write: causal-program-history (Write volatile a sop A L R W# is_{sb}) sb
shows causal-program-history is_{sb} (sb @ [$\text{Write}_{\mathsf{sb}}$ volatile a sop v A L R W])
proof
fix sb_1 sb_2
assume sb: sb @ [$\text{Write}_{\mathsf{sb}}$ volatile a sop v A L R W] = sb_1 @ sb_2
from causal-Write
interpret causal-program-history Write volatile a sop A L R W# is_{sb} sb **.**
show ∃is. instrs sb_2 @ is_{sb} = is @ prog-instrs sb_2
proof (cases sb_2)
case Nil
thus ?thesis
by simp
next
case (Cons r sb′)
from Cons-to-snoc [of r sb′] Cons **obtain** ys y **where** sb_2-snoc: sb_2=ys@[y]
by auto
with sb **obtain** y: y = $\text{Write}_{\mathsf{sb}}$ volatile a sop v A L R W **and** sb: sb = sb_1@ys
by simp

from causal-program-history [OF sb] **obtain** is **where**
instrs ys @ Write volatile a sop A L R W# is_{sb} = is @ prog-instrs ys
by auto
then show ?thesis
by (simp add: sb_2-snoc y instrs-append prog-instrs-append)
qed
qed

lemma causal-program-history-$\text{Prog}_{\mathsf{sb}}$:
assumes causal-Write: causal-program-history is_{sb} sb
shows causal-program-history (is_{sb}@mis) (sb @ [$\text{Prog}_{\mathsf{sb}}$ p_1 p_2 mis])

proof
  fix sb_1 sb_2
  assume sb: sb @ [$Prog_{sb}$ p_1 p_2 mis] = sb_1 @ sb_2
  from causal-Write
  interpret causal-program-history is_{sb} sb **.**
  show ∃is. instrs sb_2 @ (is_{sb}@mis) = is @ prog-instrs sb_2
  proof (cases sb_2)
    case Nil
    thus ?thesis
      by simp
  next
    case (Cons r sb′)
    from Cons-to-snoc [of r sb′] Cons **obtain** ys y **where** sb_2-snoc: sb_2=ys@[y]
      by auto
    with sb **obtain** y: y = $Prog_{sb}$ p_1 p_2 mis **and** sb: sb = sb_1@ys
      by simp

    from causal-program-history [OF sb] **obtain** is **where**
      instrs ys @ (is_{sb} @ mis) = is @ prog-instrs (ys@[$Prog_{sb}$ p_1 p_2 mis])
      by (auto simp add: prog-instrs-append)
    then show ?thesis
      by (simp add: sb_2-snoc y instrs-append prog-instrs-append)
  qed
qed

lemma causal-program-history-Ghost:
  assumes causal-$Ghost_{sb}$: causal-program-history (Ghost A L R W # is_{sb}) sb
  shows causal-program-history is_{sb} (sb @ [$Ghost_{sb}$ A L R W])
proof
  fix sb_1 sb_2
  assume sb: sb @ [$Ghost_{sb}$ A L R W] = sb_1 @ sb_2
  from causal-$Ghost_{sb}$
  interpret causal-program-history Ghost A L R W # is_{sb} sb **.**
  show ∃is. instrs sb_2 @ is_{sb} = is @ prog-instrs sb_2
  proof (cases sb_2)
    case Nil
    thus ?thesis
      by simp
  next
    case (Cons r sb′)
    from Cons-to-snoc [of r sb′] Cons **obtain** ys y **where** sb_2-snoc: sb_2=ys@[y]
      by auto
    with sb **obtain** y: y = $Ghost_{sb}$ A L R W **and** sb: sb = sb_1@ys
      by simp

    from causal-program-history [OF sb] **obtain** is **where**
      instrs ys @ Ghost A L R W # is_{sb} = is @ prog-instrs ys
      by auto
    then show ?thesis
      by (simp add: sb_2-snoc y instrs-append prog-instrs-append)

qed
qed

lemma hd-prog-last-prog-end: $[\![p = \text{hd-prog } p \text{ sb} ; \text{last-prog } p \text{ sb} = p_{\mathsf{sb}}]\!] \Longrightarrow p = \text{hd-prog}$ p_{sb} sb
by (induct sb) (auto split: memref.splits)

lemma hd-prog-idem: hd-prog (hd-prog p xs) xs = hd-prog p xs
by (induct xs) (auto split: memref.splits)

lemma last-prog-idem: last-prog (last-prog p sb) sb = last-prog p sb
by (induct sb) (auto split: memref.splits)

lemma last-prog-hd-prog-append:
last-prog (hd-prog p_{sb} (sb@sb$'$)) sb =last-prog (hd-prog p_{sb} sb$'$) sb
apply (induct sb)
apply (auto split: memref.splits)
done

lemma last-prog-hd-prog: last-prog (hd-prog p xs) xs = last-prog p xs
by (induct xs) (auto split: memref.splits)

lemma last-prog-append-Read$_{\mathsf{sb}}$:
$\bigwedge$p. last-prog p (sb @ [Read$_{\mathsf{sb}}$ volatile a t v]) = last-prog p sb
by (induct sb) (auto split: memref.splits)

lemma last-prog-append-Write$_{\mathsf{sb}}$:
$\bigwedge$p. last-prog p (sb @ [Write$_{\mathsf{sb}}$ volatile a sop v A L R W]) = last-prog p sb
by (induct sb) (auto split: memref.splits)

lemma last-prog-append-Prog$_{\mathsf{sb}}$:
$\bigwedge$x. last-prog x (sb@[Prog$_{\mathsf{sb}}$ p p$'$ mis]) = p$'$
by (induct sb) (auto split: memref.splits)

lemma hd-prog-append-Prog$_{\mathsf{sb}}$: hd-prog x (sb @ [Prog$_{\mathsf{sb}}$ p p$'$ mis]) = hd-prog p sb
by (induct sb) (auto split: memref.splits)

lemma hd-prog-last-prog-append-Prog$_{\mathsf{sb}}$:
$\bigwedge$p$'$. hd-prog p$'$ xs = p$'$ $\Longrightarrow$ last-prog p$'$ xs = p_1 $\Longrightarrow$
hd-prog p$'$ (xs @ [Prog$_{\mathsf{sb}}$ p_1 p_2 mis]) = p$'$
apply (induct xs)
apply (auto split: memref.splits)
done

lemma hd-prog-append-Ghost$_{\mathsf{sb}}$:

hd-prog p (sb@[$\text{Ghost}_{\mathsf{sb}}$ A R L W]) = hd-prog p sb
by (induct sb) (auto split: memref.splits)

lemma last-prog-append-$\text{Ghost}_{\mathsf{sb}}$:
$\bigwedge$p. last-prog p (sb @ [$\text{Ghost}_{\mathsf{sb}}$ A L R W]) = last-prog p sb
by (induct sb) (auto split: memref.splits)

lemma dropWhile-all-False-conv:
$\forall$x $\in$ set xs. $\neg$ P x $\Longrightarrow$ dropWhile P xs = xs
by (induct xs) auto

lemma dropWhile-append-all-False:
$\forall$y $\in$ set ys. $\neg$ P y $\Longrightarrow$
dropWhile P (xs@ys) = dropWhile P xs @ ys
apply (induct xs)
apply (auto simp add: dropWhile-all-False-conv)
done

lemma reads-consistent-append-first:
$\bigwedge$m ys. reads-consistent pending-write $\mathcal{O}$ m (xs @ ys) $\Longrightarrow$ reads-consistent pending-write $\mathcal{O}$ m xs
by (clarsimp simp add: reads-consistent-append)

lemma reads-consistent-takeWhile:
assumes consis: reads-consistent pending-write $\mathcal{O}$ m sb
shows reads-consistent pending-write $\mathcal{O}$ m (takeWhile P sb)
using reads-consistent-append [**where** xs=(takeWhile P sb) **and** ys=(dropWhile P sb)] consis
apply (simp add: reads-consistent-append)
done

lemma flush-flush-all-until-volatile-write-$\text{Write}_{\mathsf{sb}}$-volatile-commute:
$\bigwedge$i m. $[\![$i < length ts; ts!i=(p,is,xs,$\text{Write}_{\mathsf{sb}}$ True a sop v A L R W#sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
$\forall$i < length ts. ($\forall$j < length ts. i $\neq$ j $\longrightarrow$
(let (-,-,-,sb_i,-,-,-) = ts!i;
(-,-,-,sb_j,-,-,-) = ts!j
in outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ sb_i $\cap$
outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_j) = {}));
$\forall$j < length ts. i $\neq$ j $\longrightarrow$
(let (-,-,-,sb_j,-,-,-) = ts!j in a $\notin$ outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_j))$]\!]$
$\Longrightarrow$
flush (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb)
((flush-all-until-volatile-write ts m)(a := v)) =
flush-all-until-volatile-write (ts[i := (p,is,xs, sb, $\mathcal{D}'$, $\mathcal{O}'$,$\mathcal{R}'$)])
(m(a := v))
proof (induct ts)
case Nil **thus** ?case

```
    by simp
next
  case (Cons l ts)
  note i-bound =  ‹i < length (l#ts)›
  note ith = ‹(l#ts)!i = (p,is,xs,Write_sb True a sop v A L R W#sb,𝒟,𝒪,ℛ)›
  note disj = ‹∀i < length (l#ts). (∀j < length (l#ts). i ≠ j ⟶
                (let (-,-,-,sb_i,-,-,-) = (l#ts)!i;
                    (-,-,-,sb_j,-,-,-) = (l#ts)!j
                 in outstanding-refs is-Write_sb sb_i ∩
                    outstanding-refs is-Write_sb (takeWhile (Not ∘ is-volatile-Write_sb) sb_j) =
{}))›
  note a-notin = ‹∀j < length (l#ts). i ≠ j ⟶
              (let (-,-,-,sb_j,-,-,-) = (l#ts)!j
              in a ∉ outstanding-refs is-Write_sb (takeWhile (Not ∘ is-volatile-Write_sb) sb_j))›
  show ?case
  proof (cases i)
    case 0
    from ith 0 have l: l = (p,is,xs,Write_sb True a sop v A L R W#sb,𝒟,𝒪,ℛ)
      by simp
    have a-notin-ts:
      a ∉ ⋃((λ(-,-,-,sb,-,-,-). outstanding-refs is-Write_sb
                         (takeWhile (Not ∘ is-volatile-Write_sb) sb)) ` set ts) (is a ∉ ?U)
    proof
      assume a ∈ ?U
      from in-Union-image-nth-conv [OF this]
      obtain j p_j is_j 𝒪_j ℛ_j 𝒟_j xs_j sb_j where
 j-bound: j < length ts and
 jth: ts!j = (p_j,is_j,xs_j,sb_j,𝒟_j,𝒪_j,ℛ_j) and
 a-in-j: a ∈ outstanding-refs is-Write_sb (takeWhile (Not ∘ is-volatile-Write_sb) sb_j)
 by fastforce
      from a-notin [rule-format, of Suc j] j-bound 0 a-in-j
      show False
 by (auto simp add: jth)
    qed

    from a-notin-ts
    have (flush-all-until-volatile-write ts m)(a := v) =
             flush-all-until-volatile-write ts (m(a := v))
      apply −
      apply (rule update-commute′ [where F={a} and G=?U and
 g=flush-all-until-volatile-write ts])
      apply (auto intro: flush-all-until-volatile-wirte-mem-independent
             flush-all-until-volatile-write-unchanged-addresses)
      done

    moreover

    let ?SB = outstanding-refs is-Write_sb (takeWhile (Not ∘ is-volatile-Write_sb) sb)

    have U-SB-disj: ?U ∩ ?SB = {}
```

proof −
{
fix a′
assume a′-in-U: a′ ∈ ?U
have a′ ∉ ?SB
proof
assume a′-in-SB: a′ ∈ ?SB
hence a′-in-SB′: a′ ∈ outstanding-refs is-Write$_{sb}$ sb
apply (clarsimp simp add: outstanding-refs-conv)
apply (drule set-takeWhileD)
subgoal for x
apply (rule-tac x=x **in** exI)
apply (auto simp add: is-Write$_{sb}$-def split:memref.splits)
done
done
from in-Union-image-nth-conv [OF a′-in-U]
obtain j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs_j sb_j **where**
j-bound: j < length ts **and**
jth: ts!j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$) **and**
a′-in-j: a′ ∈ outstanding-refs is-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)
by fastforce

from disj [rule-format, of 0 Suc j] 0 j-bound a′-in-SB′ a′-in-j jth l
show False
by auto
qed
}
moreover
{
fix a′
assume a′-in-SB: a′ ∈ ?SB
hence a′-in-SB′: a′ ∈ outstanding-refs is-Write$_{sb}$ sb
apply (clarsimp simp add: outstanding-refs-conv)
apply (drule set-takeWhileD)
subgoal for x
apply (rule-tac x=x **in** exI)
apply (auto simp add: is-Write$_{sb}$-def split:memref.splits)
done
done
have a′ ∉ ?U
proof
assume a′ ∈ ?U
from in-Union-image-nth-conv [OF this]
obtain j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs_j sb_j **where**
j-bound: j < length ts **and**
jth: ts!j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{R}_j$,$\mathcal{O}_j$) **and**
a′-in-j: a′ ∈ outstanding-refs is-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)
by fastforce

from disj [rule-format, of 0 Suc j] j-bound a′-in-SB′ a′-in-j jth l

```
  show False
   by auto
qed
    }
    ultimately
    show ?thesis by blast
  qed

  have flush (takeWhile (Not ∘ is-volatile-Write_sb) sb)
         (flush-all-until-volatile-write ts (m(a := v))) =
        flush-all-until-volatile-write ts
         (flush (takeWhile (Not ∘ is-volatile-Write_sb) sb) (m(a := v)))
    apply (rule update-commute' [where g = flush-all-until-volatile-write ts ,
           OF - - - - U-SB-disj])
    apply (auto intro: flush-all-until-volatile-wirte-mem-independent
           flush-all-until-volatile-write-unchanged-addresses
           flush-unchanged-addresses
           flushed-values-mem-independent simp del: o-apply)
    done

  ultimately
  have flush (takeWhile (Not ∘ is-volatile-Write_sb) sb)
         ((flush-all-until-volatile-write ts m)(a := v)) =
        flush-all-until-volatile-write ts
         (flush (takeWhile (Not ∘ is-volatile-Write_sb) sb) (m(a := v)))
    by simp

  then show ?thesis
    by (auto simp add: l 0 o-def simp del: fun-upd-apply)
 next
  case (Suc n)

  obtain p_l is_l 𝒪_l ℛ_l 𝒟_j xs_l sb_l where l: l = (p_l,is_l,xs_l,sb_l,𝒟_j,𝒪_l,ℛ_l)
    by (cases l)

  from i-bound ith disj a-notin
  have
    flush (takeWhile (Not ∘ is-volatile-Write_sb) sb)
      ((flush-all-until-volatile-write ts
         (flush (takeWhile (Not ∘ is-volatile-Write_sb) sb_l) m))
        (a := v)) =
     flush-all-until-volatile-write (ts[n := (p,is, xs, sb,𝒟', 𝒪',ℛ')])
      ((flush (takeWhile (Not ∘ is-volatile-Write_sb) sb_l) m)(a := v))
    apply −
    apply (rule Cons.hyps)
    apply (force simp add: Suc Let-def simp del: o-apply)+
    done

  moreover
```

let ?SB = outstanding-refs is-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_l$)
have a ∉ ?SB
proof
assume a ∈ ?SB
with a-notin [rule-format, of 0]
show False
by (auto simp add: l Suc)
qed
then
have ((flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_l$) m)(a := v)) =
(flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_l$) (m(a := v)))
apply −
apply (rule update-commute′ [**where** m=m **and** F={a} **and** G=?SB])
apply (auto intro:
flush-unchanged-addresses
flushed-values-mem-independent simp del: o-apply)
done

ultimately
show ?thesis
by (simp add: l Suc del: fun-upd-apply o-apply)
qed
qed

lemma (**in** program)
⋀sb′ p. history-consistent j (hd-prog p (sb@sb′)) (sb@sb′) ⟹
last-prog p (sb@sb′) = p ⟹
last-prog (hd-prog p (sb@sb′)) sb = hd-prog p sb′
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons r sb$_1$)
have consis: history-consistent j (hd-prog p ((r # sb$_1$) @ sb′)) ((r # sb$_1$) @ sb′)
by fact
have last-prog: last-prog p ((r # sb$_1$) @ sb′) = p **by** fact
show ?case
proof (cases r)
case Write$_{sb}$ **with** Cons **show** ?thesis **by** auto
next
case Read$_{sb}$ **with** Cons **show** ?thesis **by** (auto split: option.splits)
next
case (Prog$_{sb}$ p$_1$ p$_2$ is)
from last-prog **have** last-prog-p$_2$: last-prog p$_2$ (sb$_1$ @ sb′) = p

```
      by (simp add: Progsb)
    from consis obtain consis′: history-consistent j p2 (sb1 @ sb′)
      by (simp add: Progsb)

    hence history-consistent j (hd-prog p2 (sb1 @ sb′)) (sb1 @ sb′)
      by (rule history-consistent-hd-prog)
    from Cons.hyps [OF this ]
    have last-prog p2 sb1 = hd-prog p sb′
      oops

lemma last-prog-to-last-prog-same: ⋀p′. last-prog p′ sb = p ⟹ last-prog p sb = p
  by (induct sb) (auto split: memref.splits)

lemma last-prog-hd-prog-same: ⟦last-prog p′ sb = p; hd-prog p′ sb = p′⟧ ⟹ hd-prog p
sb = p′
  by (induct sb) (auto split : memref.splits)

lemma last-prog-hd-prog-last-prog:
  last-prog p′ (sb@sb′) = p ⟹ hd-prog p′ (sb@sb′) = p′ ⟹
   last-prog (hd-prog p sb′) sb = last-prog p′ sb
apply (induct sb)
apply (simp add: last-prog-hd-prog-same)
apply (auto split : memref.splits)
done

lemma (in program) last-prog-hd-prog-append′:
⋀sb′ p. history-consistent j (hd-prog p (sb@sb′)) (sb@sb′) ⟹
          last-prog p (sb@sb′) = p ⟹
 last-prog (hd-prog p sb′) sb = hd-prog p sb′
proof (induct sb)
  case Nil thus ?case by simp
next
  case (Cons r sb1)
  have consis: history-consistent j (hd-prog p ((r # sb1) @ sb′)) ((r # sb1) @ sb′)
    by fact
  have last-prog: last-prog p ((r # sb1) @ sb′) = p by fact
  show ?case
  proof (cases r)
    case Writesb with Cons show ?thesis by auto
  next
    case Readsb with Cons show ?thesis by (auto split: option.splits)
  next
    case (Progsb p1 p2 is)
    from last-prog have last-prog-p2: last-prog p2 (sb1 @ sb′) = p
      by (simp add: Progsb)
    from last-prog-to-last-prog-same [OF this]
    have last-prog-p: last-prog p (sb1 @ sb′) = p.
    from consis obtain consis′: history-consistent j p2 (sb1 @ sb′)
      by (simp add: Progsb)
    from history-consistent-hd-prog-p [OF consis′]
```

have hd-prog-p_2: hd-prog p_2 (sb_1 @ sb′) = p_2 **by** simp
from consis′ **have** history-consistent j (hd-prog p (sb_1 @ sb′)) (sb_1 @ sb′)
by (rule history-consistent-hd-prog)
from Cons.hyps [OF this last-prog-p]
have last-prog (hd-prog p sb′) sb_1 = hd-prog p sb′.
moreover
from last-prog-hd-prog-last-prog [OF last-prog-p_2 hd-prog-p_2]
have last-prog (hd-prog p sb′) sb_1 = last-prog p_2 sb_1.
ultimately
have last-prog p_2 sb_1 = hd-prog p sb′
by simp
thus ?thesis
by (simp add: $\text{Prog}_{\mathsf{sb}}$)
next
case $\text{Ghost}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** (auto split: option.splits)
qed
qed

lemma flush-all-until-volatile-write-$\text{Write}_{\mathsf{sb}}$-non-volatile-commute:
⋀i m. ⟦i < length ts; ts!i=(p,is,xs,$\text{Write}_{\mathsf{sb}}$ False a sop v A L R W#sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
∀ i < length ts. (∀ j < length ts. i ≠ j ⟶
(let (-,-,-,sb_i,-,-,-) = ts!i;
(-,-,-,sb_j,-,-,-) = ts!j
in outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ sb_i ∩
outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_j) =
{}));
∀ j < length ts. i ≠ j ⟶
(let (-,-,-,sb_j,-,-,-) = ts!j in a ∉ outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ (takeWhile (Not ∘
is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_j))⟧
⟹ flush-all-until-volatile-write (ts[i := (p,is, xs, sb,$\mathcal{D}'$, $\mathcal{O}$,$\mathcal{R}'$)])(m(a := v)) =
flush-all-until-volatile-write ts m
proof (induct ts)
case Nil **thus** ?case
by simp
next
case (Cons l ts)
note i-bound = ‹i < length (l#ts)›
note ith = ‹(l#ts)!i = (p,is,xs,$\text{Write}_{\mathsf{sb}}$ False a sop v A L R W#sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)›
note disj = ‹∀ i < length (l#ts). (∀ j < length (l#ts). i ≠ j ⟶
(let (-,-,-,sb_i,-,-,-) = (l#ts)!i;
(-,-,-,sb_j,-,-,-) = (l#ts)!j
in outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ sb_i ∩
outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_j) =
{}))›
note a-notin = ‹∀ j < length (l#ts). i ≠ j ⟶
(let (-,-,-,sb_j,-,-,-) = (l#ts)!j
in a ∉ outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_j))›
show ?case
proof (cases i)
case 0

```
    from ith 0 have l: l = (p,is,xs,Write_sb False a sop v A L R W#sb,𝒟,𝒪,ℛ)
      by simp
    thus ?thesis
      by (simp add: 0 del: fun-upd-apply)
  next
    case (Suc n)
    obtain p_l is_l 𝒪_l ℛ_l 𝒟_l xs_l sb_l where l: l = (p_l,is_l,xs_l,sb_l,𝒟_l,𝒪_l,ℛ_l)
      by (cases l)

    from i-bound ith disj a-notin
    have
      flush-all-until-volatile-write (ts[n := (p,is,xs, sb, 𝒟', 𝒪,ℛ')])
          ((flush (takeWhile (Not ∘ is-volatile-Write_sb) sb_l) m)(a := v)) =
       flush-all-until-volatile-write ts
          (flush (takeWhile (Not ∘ is-volatile-Write_sb) sb_l) m)
      apply −
      apply (rule Cons.hyps)
      apply (force simp add: Suc Let-def simp del: o-apply)+
      done

    moreover

    let ?SB = outstanding-refs is-Write_sb (takeWhile (Not ∘ is-volatile-Write_sb) sb_l)
    have a ∉ ?SB
    proof
      assume a ∈ ?SB
      with a-notin [rule-format, of 0]
      show False
 by (auto simp add: l Suc)
    qed
    then
    have ((flush (takeWhile (Not ∘ is-volatile-Write_sb) sb_l) m)(a := v)) =
          (flush (takeWhile (Not ∘ is-volatile-Write_sb) sb_l) (m(a := v)))
      apply −
      apply (rule update-commute' [where m=m and F={a} and G=?SB])
      apply (auto intro:
             flush-unchanged-addresses
             flushed-values-mem-independent simp del: o-apply)
      done

    ultimately
    show ?thesis
      by (simp add: l Suc del: fun-upd-apply o-apply)
  qed
qed

lemma (in program) history-consistent-access-last-read':
  ⋀p. history-consistent j p (sb @ [Read_sb volatile a t v]) ⟹
       j t = Some v
apply (induct sb)
```

apply (auto split: memref.splits option.splits)
done

lemma (**in** program) history-consistent-access-last-read:
history-consistent j p (rev ($\text{Read}_{\mathsf{sb}}$ volatile a t v # sb)) $\Longrightarrow$ j t = Some v
by (simp add: history-consistent-access-last-read$'$)

lemma flush-all-until-volatile-write-$\text{Read}_{\mathsf{sb}}$-commute:
$\bigwedge$i m. $[\![$i < length ts; ts!i=(p,is,j,$\text{Read}_{\mathsf{sb}}$ volatile a t v#sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)$]\!]$
$\Longrightarrow$ flush-all-until-volatile-write (ts[i := (p,is,j, sb, $\mathcal{D}'$, $\mathcal{O}$,$\mathcal{R}'$)]) m
= flush-all-until-volatile-write ts m
proof (induct ts)
case Nil **thus** ?case
by simp
next
case (Cons l ts)
note i-bound = ‹i < length (l#ts)›
note ith = ‹(l#ts)!i = (p,is,j,$\text{Read}_{\mathsf{sb}}$ volatile a t v#sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)›
show ?case
proof (cases i)
case 0
from ith 0 **have** l: l = (p,is,j,$\text{Read}_{\mathsf{sb}}$ volatile a t v#sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by simp
thus ?thesis
by (simp add: 0 del: fun-upd-apply)
next
case (Suc n)
obtain p_l is_l $\mathcal{O}_l$ $\mathcal{R}_l$ $\mathcal{D}_l$ j_l sb_l **where** l: l = (p_l,is_l,j_l,sb_l,$\mathcal{D}_l$,$\mathcal{O}_l$,$\mathcal{R}_l$)
by (cases l)

from i-bound ith
have flush-all-until-volatile-write (ts[n := (p,is,j, sb, $\mathcal{D}'$, $\mathcal{O}$,$\mathcal{R}'$)])
(flush (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_l) m) =
flush-all-until-volatile-write ts
(flush (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_l) m)
apply −
apply (rule Cons.hyps)
apply (auto simp add: Suc l)
done

then show ?thesis
by (simp add: Suc l)
qed
qed

lemma flush-all-until-volatile-write-$\text{Ghost}_{\mathsf{sb}}$-commute:
$\bigwedge$i m. $[\![$i < length ts; ts!i=(p,is,j,$\text{Ghost}_{\mathsf{sb}}$ A L R W#sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)$]\!]$
$\Longrightarrow$ flush-all-until-volatile-write (ts[i := (p$'$,is$'$,j$'$, sb, $\mathcal{D}'$, $\mathcal{O}'$,$\mathcal{R}'$)]) m
= flush-all-until-volatile-write ts m
proof (induct ts)

case Nil **thus** ?case
  by simp
next
  case (Cons l ts)
  note i-bound = ‹i < length (l#ts)›
  note ith = ‹(l#ts)!i = (p,is,j,Ghost$_{sb}$ A L R W#sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)›
  show ?case
  proof (cases i)
    case 0
    from ith 0 **have** l: l = (p,is,j,Ghost$_{sb}$ A L R W#sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
      by simp
    thus ?thesis
      by (simp add: 0 del: fun-upd-apply)
  next
    case (Suc n)
    obtain p_l is_l $\mathcal{O}_l$ $\mathcal{R}_l$ $\mathcal{D}_l$ j_l sb_l **where** l: l = (p_l,is_l,j_l,sb_l,$\mathcal{D}_l$,$\mathcal{O}_l$,$\mathcal{R}_l$)
      by (cases l)

    from i-bound ith
    have flush-all-until-volatile-write (ts[n := (p′,is′,j′, sb, $\mathcal{D}'$, $\mathcal{O}'$,$\mathcal{R}'$)])
          (flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_l) m) =
        flush-all-until-volatile-write ts
          (flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_l) m)
      apply −
      apply (rule Cons.hyps)
      apply (auto simp add: Suc l)
      done

    then show ?thesis
      by (simp add: Suc l)
  qed
qed

lemma flush-all-until-volatile-write-Prog$_{sb}$-commute:
  ⋀i m. ⟦i < length ts; ts!i=(p,is,j,Prog$_{sb}$ p_1 p_2 mis#sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)⟧
     ⟹ flush-all-until-volatile-write (ts[i := (p,is, j, sb,$\mathcal{D}'$, $\mathcal{O}$,$\mathcal{R}'$)]) m
     = flush-all-until-volatile-write ts m
proof (induct ts)
  case Nil **thus** ?case
    by simp
next
  case (Cons l ts)
  note i-bound = ‹i < length (l#ts)›
  note ith = ‹(l#ts)!i = (p,is,j,Prog$_{sb}$ p_1 p_2 mis#sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)›
  show ?case
  proof (cases i)
    case 0
    from ith 0 **have** l: l = (p,is,j,Prog$_{sb}$ p_1 p_2 mis#sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
      by simp
    thus ?thesis

by (simp add: 0 del: fun-upd-apply)
next
case (Suc n)
obtain p_l is_l $\mathcal{O}_l$ $\mathcal{R}_l$ $\mathcal{D}_l$ j_l sb_l **where** l: l = (p_l,is_l,j_l,sb_l,$\mathcal{D}_l$,$\mathcal{O}_l$,$\mathcal{R}_l$)
by (cases l)

from i-bound ith
have flush-all-until-volatile-write (ts[n := (p,is, j, sb,$\mathcal{D}'$, $\mathcal{O}$,$\mathcal{R}'$)])
(flush (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_l) m) =
flush-all-until-volatile-write ts
(flush (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_l) m)
apply −
apply (rule Cons.hyps)
apply (auto simp add: Suc l)
done

then show ?thesis
by (simp add: Suc l)
qed
qed

lemma flush-all-until-volatile-write-append-Prog$_{sb}$-commute:
$\bigwedge$i m. $[\![$i < length ts; ts!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)$]\!]$
$\Longrightarrow$ flush-all-until-volatile-write (ts[i := (p_2,is@mis, j, sb@[Prog$_{sb}$ p_1 p_2 mis],$\mathcal{D}'$, $\mathcal{O}$,$\mathcal{R}'$)]) m
= flush-all-until-volatile-write ts m
proof (induct ts)
case Nil **thus** ?case
by simp
next
case (Cons l ts)
note i-bound = ‹i < length (l#ts)›
note ith = ‹(l#ts)!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)›
show ?case
proof (cases i)
case 0
from ith 0 **have** l: l = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by simp
thus ?thesis
by (simp add: 0 flush-append-Prog$_{sb}$ del: fun-upd-apply)
next
case (Suc n)
obtain p_l is_l $\mathcal{O}_l$ $\mathcal{R}_l$ $\mathcal{D}_l$ j_l sb_l **where** l: l = (p_l,is_l,j_l,sb_l,$\mathcal{D}_l$,$\mathcal{O}_l$,$\mathcal{R}_l$)
by (cases l)

from i-bound ith
have flush-all-until-volatile-write
(ts[n := (p_2,is@mis,j, sb@[Prog$_{sb}$ p_1 p_2 mis], $\mathcal{D}'$, $\mathcal{O}$,$\mathcal{R}'$)])
(flush (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_l) m) =

```
        flush-all-until-volatile-write ts
          (flush (takeWhile (Not ∘ is-volatile-Write_sb) sb_l) m)
      apply −
      apply (rule Cons.hyps)
      apply (auto simp add: Suc l)
      done

    then show ?thesis
      by (simp add: Suc l)
  qed
qed
```

```
lemma (in program) history-consistent-append-Prog_sb:
  assumes step: j⊢ p →_p (p′, mis)
  shows history-consistent j (hd-prog p xs) xs ⟹ last-prog p xs = p ⟹
       history-consistent j (hd-prog p′ (xs@[Prog_sb p p′ mis])) (xs@[Prog_sb p p′ mis])
proof (induct xs)
  case Nil with step show ?case by simp
next
  case (Cons x xs)
  note consis = ‹history-consistent j (hd-prog p (x # xs)) (x # xs)›
  note last = ‹last-prog p (x#xs) = p›
  show ?case
  proof (cases x)
    case Write_sb with Cons show ?thesis by (auto simp add: read-tmps-append)
  next
    case Read_sb with Cons show ?thesis by (auto split: option.splits)
  next
    case (Prog_sb p_1 p_2 mis′)
    from consis obtain
      step: j |‘(dom j − read-tmps (xs @ [Prog_sb p p′ mis]))⊢ p_1 →_p (p_2, mis′) and
      consis′: history-consistent j p_2 xs
      by (auto simp add: Prog_sb read-tmps-append)
    from last have last-p_2: last-prog p_2 xs = p
      by (simp add: Prog_sb)
    from last-prog-to-last-prog-same [OF this]
    have last-prog′: last-prog p xs = p.
    from history-consistent-hd-prog [OF consis′]
    have consis″: history-consistent j (hd-prog p xs) xs.
    from Cons.hyps [OF this last-prog′]
    have history-consistent j (hd-prog p′ (xs @ [Prog_sb p p′ mis]))
          (xs @ [Prog_sb p p′ mis]).
    from history-consistent-hd-prog [OF this]
    have history-consistent j (hd-prog p_2 (xs @ [Prog_sb p p′ mis]))
         (xs @ [Prog_sb p p′ mis]).
    moreover
    from history-consistent-hd-prog-p [OF consis′]
```

have p_2 = hd-prog p_2 xs.
from hd-prog-last-prog-append-$Prog_{sb}$ [OF this [symmetric] last-p_2]
have hd-prog p_2 (xs @ [$Prog_{sb}$ p p′ mis]) = p_2
by simp
ultimately
have history-consistent j p_2 (xs @ [$Prog_{sb}$ p p′ mis])
by simp
thus ?thesis
by (simp add: $Prog_{sb}$ step)
next
case $Ghost_{sb}$ **with** Cons **show** ?thesis **by** (auto)
qed
qed

primrec release :: ′a memref list ⇒ addr set ⇒ rels ⇒ rels
where
release [] S $\mathcal{R}$ = $\mathcal{R}$
| release (x#xs) S $\mathcal{R}$ =
(case x of
$Write_{sb}$ volatile - - - A L R W ⇒
(if volatile then release xs (S ∪ R − L) Map.empty
else release xs S $\mathcal{R}$)
| $Ghost_{sb}$ A L R W ⇒ release xs (S ∪ R − L) (augment-rels S R $\mathcal{R}$)
| - ⇒ release xs S $\mathcal{R}$)

lemma augment-rels-shared-exchange: ∀a ∈ R. (a ∈ S′) = (a ∈ S) ⟹ augment-rels S R $\mathcal{R}$ = augment-rels S′ R $\mathcal{R}$
apply (rule ext)
apply (auto simp add: augment-rels-def split: option.splits)
done

lemma sharing-consistent-shared-exchange:
assumes shared-eq: ∀a ∈ all-acquired sb. $\mathcal{S}'$ a = $\mathcal{S}$ a
assumes consis: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb
shows sharing-consistent $\mathcal{S}'$ $\mathcal{O}$ sb
using shared-eq consis
proof (induct sb arbitrary: $\mathcal{S}$ $\mathcal{S}'$ $\mathcal{O}$)
case Nil **thus** ?case **by** auto
next
case (Cons x sb)
show ?case
proof (cases x)
case ($Write_{sb}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)

case True

from Cons.prems **obtain**
A-shared-owns: A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ $\cup$ A − R) sb **and**
shared-eq: $\forall$a $\in$ A $\cup$ all-acquired sb. $\mathcal{S}'$ a = $\mathcal{S}$ a
by (clarsimp simp add: Write$_{\mathsf{sb}}$ True)
from shared-eq
have shared-eq′: $\forall$a$\in$ all-acquired sb. ($\mathcal{S}'$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a
by (auto simp add: augment-shared-def restrict-shared-def)
from Cons.hyps [OF shared-eq′ consis′]
have sharing-consistent ($\mathcal{S}'$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ $\cup$ A − R) sb**.**
thus ?thesis
using A-shared-owns L-A A-R R-owns shared-eq
by (auto simp add: Write$_{\mathsf{sb}}$ True domIff)
next
case False **with** Cons **show** ?thesis
by (auto simp add: Write$_{\mathsf{sb}}$)
qed
next
case Read$_{\mathsf{sb}}$ **with** Cons **show** ?thesis
by auto
next
case Prog$_{\mathsf{sb}}$ **with** Cons **show** ?thesis
by auto
next
case (Ghost$_{\mathsf{sb}}$ A L R W)
from Cons.prems **obtain**
A-shared-owns: A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ $\cup$ A − R) sb **and**
shared-eq: $\forall$a $\in$ A $\cup$ all-acquired sb. $\mathcal{S}'$ a = $\mathcal{S}$ a
by (clarsimp simp add: Ghost$_{\mathsf{sb}}$)
from shared-eq
have shared-eq′: $\forall$a$\in$all-acquired sb. ($\mathcal{S}'$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a
by (auto simp add: augment-shared-def restrict-shared-def)
from Cons.hyps [OF shared-eq′ consis′]
have sharing-consistent ($\mathcal{S}'$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ $\cup$ A − R) sb**.**
thus ?thesis
using A-shared-owns L-A A-R R-owns shared-eq
by (auto simp add: Ghost$_{\mathsf{sb}}$ domIff)
qed
qed

lemma release-shared-exchange:
assumes shared-eq: $\forall$a $\in$ $\mathcal{O}$ $\cup$ all-acquired sb. $\mathcal{S}'$ a = $\mathcal{S}$ a
assumes consis: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb

shows release sb (dom $\mathcal{S}'$) $\mathcal{R}$ = release sb (dom $\mathcal{S}$) $\mathcal{R}$
using shared-eq consis
proof (induct sb arbitrary: $\mathcal{S}$ $\mathcal{S}'$ $\mathcal{O}$ $\mathcal{R}$)
  case Nil **thus** ?case **by** auto
next
  case (Cons x sb)
  show ?case
  proof (cases x)
    case (Write$_{sb}$ volatile a sop v A L R W)
    show ?thesis
    proof (cases volatile)
      case True

      from Cons.prems **obtain**
 A-shared-owns: A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and**
 consis′: sharing-consistent ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) ($\mathcal{O}$ $\cup$ A − R) sb **and**
        shared-eq: $\forall$ a $\in$ $\mathcal{O}$ $\cup$ A $\cup$ all-acquired sb. $\mathcal{S}'$ a = $\mathcal{S}$ a
 by (clarsimp simp add: Write$_{sb}$ True)
      from shared-eq
      have shared-eq′: $\forall$a$\in$$\mathcal{O}$ $\cup$ A − R $\cup$ all-acquired sb. ($\mathcal{S}'$ $\oplus_W$ R $\ominus_A$ L) a = ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) a
        by (auto simp add: augment-shared-def restrict-shared-def)
      from Cons.hyps [OF shared-eq′ consis′]
      have release sb (dom ($\mathcal{S}'$ $\oplus_W$ R $\ominus_A$ L)) Map.empty = release sb (dom ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)) Map.empty **.**
      then show ?thesis
        by (auto simp add: Write$_{sb}$ True domIff)
    next
      case False **with** Cons **show** ?thesis
 by (auto simp add: Write$_{sb}$)
    qed
  next
    case Read$_{sb}$ **with** Cons **show** ?thesis
      by auto
  next
    case Prog$_{sb}$ **with** Cons **show** ?thesis
      by auto
  next
    case (Ghost$_{sb}$ A L R W)
    from Cons.prems **obtain**
     A-shared-owns: A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and**
      consis′: sharing-consistent ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) ($\mathcal{O}$ $\cup$ A − R) sb **and**
      shared-eq: $\forall$ a $\in$ $\mathcal{O}$ $\cup$ A $\cup$ all-acquired sb. $\mathcal{S}'$ a = $\mathcal{S}$ a
      by (clarsimp simp add: Ghost$_{sb}$)
    from shared-eq
    have shared-eq′: $\forall$a$\in$$\mathcal{O}$ $\cup$ A − R $\cup$ all-acquired sb. ($\mathcal{S}'$ $\oplus_W$ R $\ominus_A$ L) a = ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) a
      by (auto simp add: augment-shared-def restrict-shared-def)

from A-shared-owns shared-eq R-owns **have** $\forall$ a$\in$R. (a $\in$ dom $\mathcal{S}$) = (a $\in$ dom $\mathcal{S}'$)
by (auto simp add: domIff)
from augment-rels-shared-exchange [OF this]
have (augment-rels (dom $\mathcal{S}'$) R $\mathcal{R}$) = (augment-rels (dom $\mathcal{S}$) R $\mathcal{R}$).

with Cons.hyps [OF shared-eq′ consis′]
have release sb (dom ($\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) (augment-rels (dom $\mathcal{S}'$) R $\mathcal{R}$) =
release sb (dom ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) (augment-rels (dom $\mathcal{S}$) R $\mathcal{R}$) **by** simp
then show ?thesis
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$ domIff)
qed
qed

lemma release-append:
$\bigwedge\mathcal{S}$ $\mathcal{R}$. release (sb@xs) (dom $\mathcal{S}$) $\mathcal{R}$ = release xs (dom (share sb $\mathcal{S}$)) (release sb (dom ($\mathcal{S}$)) $\mathcal{R}$)
proof (induct sb)
case Nil **thus** ?case **by** auto
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)
case True
from Cons.hyps [of ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) Map.empty]
show ?thesis
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ True)
next
case False **with** Cons **show** ?thesis **by** (auto simp add: $\text{Write}_{\mathsf{sb}}$)
qed
next
case $\text{Read}_{\mathsf{sb}}$ **with** Cons **show** ?thesis
by auto
next
case $\text{Prog}_{\mathsf{sb}}$ **with** Cons **show** ?thesis
by auto
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
with Cons.hyps [of ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) augment-rels (dom $\mathcal{S}$) R $\mathcal{R}$]
show ?thesis
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
qed
qed

locale xvalid-program = valid-program +
fixes valid
assumes valid-implies-valid-prog:
$[\![$i < length ts;

ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$); valid ts⟧ $\Longrightarrow$ valid-prog p

assumes valid-implies-valid-prog-hd:
⟦i < length ts;
ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$); valid ts⟧ $\Longrightarrow$ valid-prog (hd-prog p sb)

assumes distinct-load-tmps-prog-step:
⟦i < length ts;
ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$); j$\vdash$p $\rightarrow_{\mathsf{p}}$ (p′,is′); valid ts⟧
$\Longrightarrow$
distinct-load-tmps is′ $\wedge$
(load-tmps is′ $\cap$ load-tmps is = {}) $\wedge$
(load-tmps is′ $\cap$ read-tmps sb) = {}

assumes valid-data-dependency-prog-step:
⟦i < length ts;
ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$); j$\vdash$p $\rightarrow_{\mathsf{p}}$ (p′,is′); valid ts⟧
$\Longrightarrow$
data-dependency-consistent-instrs (dom j $\cup$ load-tmps is) is′ $\wedge$
load-tmps is′ $\cap$ $\bigcup$(fst ‘ store-sops is) = {} $\wedge$
load-tmps is′ $\cap$ $\bigcup$(fst ‘ write-sops sb) = {}

assumes load-tmps-fresh-prog-step:
⟦i < length ts;
ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$); j$\vdash$p $\rightarrow_{\mathsf{p}}$ (p′,is′); valid ts⟧
$\Longrightarrow$
load-tmps is′ $\cap$ dom j = {}

assumes valid-sops-prog-step:
⟦j$\vdash$p $\rightarrow_{\mathsf{p}}$ (p′,is′); valid-prog p⟧$\Longrightarrow$ $\forall$sop$\in$store-sops is′. valid-sop sop

assumes prog-step-preserves-valid:
⟦i < length ts;
ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$); j$\vdash$p $\rightarrow_{\mathsf{p}}$ (p′,is′); valid ts⟧ $\Longrightarrow$
valid (ts[i:=(p′,is@is′,j,sb@[Prog$_{\mathsf{sb}}$ p p′ is′],$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])

assumes flush-step-preserves-valid:
⟦i < length ts;
ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$); (m,sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}$ (m′,sb′,$\mathcal{O}$′,$\mathcal{R}$′,$\mathcal{S}$′); valid ts⟧ $\Longrightarrow$
valid (ts[i:=(p,is,j,sb′,$\mathcal{D}$,$\mathcal{O}$′,$\mathcal{R}$′)])

assumes sbh-step-preserves-valid:
⟦i < length ts;
ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
(is,j,sb,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$ (is′,j′,sb′,m′,$\mathcal{D}$′,$\mathcal{O}$′,$\mathcal{R}$′,$\mathcal{S}$′);
valid ts⟧
$\Longrightarrow$
valid (ts[i:=(p,is′,j′,sb′,$\mathcal{D}$′,$\mathcal{O}$′,$\mathcal{R}$′)])

lemma refl': $x = y \Longrightarrow r\hat{}** x y$
  by auto

lemma no-volatile-Read$_{sb}$-volatile-reads-consistent:
  $\bigwedge$m. outstanding-refs is-volatile-Read$_{sb}$ sb = {} $\Longrightarrow$ volatile-reads-consistent m sb
  apply (induct sb)
  apply simp
  subgoal for a sb m
  apply (case-tac a)
  apply (auto split: if-split-asm)
  done
  done

theorem (**in** program) flush-store-buffer-append:
shows $\bigwedge$ts p m j $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ $\mathcal{S}$ is $\mathcal{O}'$.
 $[\![$i < length ts;
  instrs (sb@sb') @ is$_{sb}$ = is @ prog-instrs (sb@sb');
  causal-program-history is$_{sb}$ (sb@sb');
  ts!i = (p,is,j |` (dom j − read-tmps (sb@sb')),x,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
  p=hd-prog p$_{sb}$ (sb@sb');
  (last-prog p$_{sb}$ (sb@sb')) = p$_{sb}$;
  reads-consistent True $\mathcal{O}'$ m sb;
  history-consistent j p (sb@sb');
  $\forall$sop $\in$ write-sops sb. valid-sop sop;
  distinct-read-tmps (sb@sb');
  volatile-reads-consistent m sb
$]\!]$
  $\Longrightarrow$
  $\exists$is'. instrs sb' @ is$_{sb}$ = is' @ prog-instrs sb' $\wedge$
    (ts,m,$\mathcal{S}$) $\Rightarrow_{d}^{*}$
    (ts[i:=(last-prog (hd-prog p$_{sb}$ sb') sb,is',j|` (dom j − read-tmps sb'),x,
      ($\mathcal{D}$ $\vee$ outstanding-refs is-volatile-Write$_{sb}$ sb $\neq$ {}),
      acquired True sb $\mathcal{O}$, release sb (dom $\mathcal{S}$) $\mathcal{R}$)], flush sb m,share sb $\mathcal{S}$)
proof (induct sb)
  case Nil
  thus ?case **by** (auto simp add: list-update-id' split: if-split-asm)
next
case (Cons r sb)
  interpret direct-computation:
    computation direct-memop-step empty-storebuffer-step program-step λp p' is sb. sb**.**
  have ts-i:
    ts!i = (p,is,j |` (dom j − read-tmps ((r#sb)@sb')),x,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
    by fact
  have is: instrs ((r # sb) @ sb') @ is$_{sb}$ = is @ prog-instrs ((r # sb) @ sb') **by** fact

  have i-bound: i < length ts **by** fact

have causal: causal-program-history is_{sb} ((r # sb) @ sb′) **by** fact
hence causal′: causal-program-history is_{sb} (sb @ sb′)
  by (auto simp add: causal-program-history-def)

note reads-consis = ‹reads-consistent True $\mathcal{O}'$ m (r#sb)›
note p = ‹p = hd-prog p_{sb} ((r#sb)@sb′)›
note p_{sb} = ‹last-prog p_{sb} ((r # sb) @ sb′) = p_{sb}›
note hist-consis = ‹history-consistent j p ((r#sb)@sb′)›
note valid-sops = ‹$\forall$sop $\in$ write-sops (r#sb). valid-sop sop›
note dist = ‹distinct-read-tmps ((r#sb)@sb′)›
note vol-read-consis = ‹volatile-reads-consistent m (r#sb)›

show ?case
proof (cases r)
  case ($Prog_{sb}$ p_1 p_2 pis)

  from vol-read-consis
  have vol-read-consis′: volatile-reads-consistent m sb
    by (auto simp add: $Prog_{sb}$)

  from hist-consis **obtain**
    prog-step: j|‘ (dom j − read-tmps (sb @ sb′)) $\vdash$ p_1 $\to_p$ (p_2, pis) **and**
    hist-consis′: history-consistent j p_2 (sb @ sb′)
    by (auto simp add: $Prog_{sb}$)
  from p **obtain** p: p = p_1
    by (simp add: $Prog_{sb}$)

  from history-consistent-hd-prog [OF hist-consis′]
  have hist-consis″: history-consistent j (hd-prog p_2 (sb @ sb′)) (sb @ sb′) **.**
  from is
  have is: instrs (sb @ sb′) @ is_{sb} = (is @ pis) @ prog-instrs (sb @ sb′)
    by (simp add: $Prog_{sb}$)

  from ts-i is **have**
    ts-i: ts!i = (p, is,j |‘ (dom j − read-tmps (sb @ sb′)), x, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)
    by (simp add: $Prog_{sb}$)

  let ?ts′= ts[i:=(p_2,is@pis,j |‘ (dom j − read-tmps (sb @ sb′)), x,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)]
  from direct-computation.Program [OF i-bound ts-i prog-step [simplified p[symmetric]]]
  have (ts,m,$\mathcal{S}$) $\Rightarrow_d$ (?ts′,m,$\mathcal{S}$) **by** simp

  also
  from i-bound **have** i-bound′: i < length ?ts′
    by auto

  from i-bound
  have ts′-i: ?ts′!i = (p_2,is@pis,(j |‘ (dom j − read-tmps (sb @ sb′))),x, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)
    by auto

from history-consistent-hd-prog-p [OF hist-consis′]
have p_2-hd-prog: p_2 = hd-prog p_2 (sb @ sb′)**.**

from reads-consis **have** reads-consis′: reads-consistent True $\mathcal{O}'$ m sb
  by (simp add: $\mathsf{Prog_{sb}}$)

from valid-sops **have** valid-sops′: ∀sop ∈ write-sops sb. valid-sop sop
  by (simp add: $\mathsf{Prog_{sb}}$)

from dist **have** dist′: distinct-read-tmps (sb@sb′)
  by (simp add: $\mathsf{Prog_{sb}}$)

from p_{sb} **have** last-prog-p_2: last-prog p_2 (sb @ sb′) = p_{sb}
  by (simp add: $\mathsf{Prog_{sb}}$)
from hd-prog-last-prog-end [OF p_2-hd-prog this]
have p_2-hd-prog′: p_2 = hd-prog p_{sb} (sb @ sb′)**.**
from last-prog-p_2 [symmetric] **have** last-prog′: last-prog p_{sb} (sb @ sb′) = p_{sb}
  by (simp add: last-prog-idem)

from Cons.hyps [OF i-bound′ is causal′ ts′-i p_2-hd-prog′ last-prog′ reads-consis′
  hist-consis′ valid-sops′ dist′ vol-read-consis′] i-bound
obtain is′ **where**
  is′: instrs sb′ @ is_{sb} = is′ @ prog-instrs sb′ **and**
  step: (?ts′, m,$\mathcal{S}$) $\Rightarrow_{\mathsf{d}}{}^*$
    (ts[i := (last-prog (hd-prog p_{sb} sb′) sb, is′,
     j |‘ (dom j − read-tmps sb′), x, $\mathcal{D}$ ∨ outstanding-refs is-volatile-$\mathsf{Write_{sb}}$ sb ≠ {},
     acquired True sb $\mathcal{O}$,release sb (dom $\mathcal{S}$) $\mathcal{R}$)],
     flush sb m,share sb $\mathcal{S}$)
  by (auto)
from p_2-hd-prog′
have last-prog-eq: last-prog (hd-prog p_{sb} sb′) sb = last-prog p_2 sb
  by (simp add: last-prog-hd-prog-append)
note step
finally show ?thesis
  using is′
  by (simp add: $\mathsf{Prog_{sb}}$ last-prog-eq)
next
case ($\mathsf{Write_{sb}}$ volatile a sop v A L R W)
obtain D f **where** sop: sop=(D,f)
  by (cases sop)

from vol-read-consis
have vol-read-consis′: volatile-reads-consistent (m(a:=v)) sb
  by (auto simp add: $\mathsf{Write_{sb}}$)

from hist-consis **obtain**
  D-tmps: D ⊆ dom j **and**

f-v: f j = v **and**
dep: D ∩ read-tmps (sb@sb′) = {} **and**
hist-consis′: history-consistent j p (sb@sb′)
by (simp add: $\text{Write}_{\mathsf{sb}}$ sop split: option.splits)

from dist **have** dist′: distinct-read-tmps (sb@sb′) **by** (auto simp add: $\text{Write}_{\mathsf{sb}}$)

from valid-sops **obtain** valid-sop sop **and**
valid-sops′: ∀ sop ∈ write-sops sb. valid-sop sop
by (simp add: $\text{Write}_{\mathsf{sb}}$)
interpret valid-sop sop **by** fact
from valid-sop [OF sop D-tmps]
have f j = f (j |‘ D) **.**
moreover
from dep D-tmps **have** D-subset: D ⊆ (dom j − read-tmps (sb@sb′))
by auto
moreover from D-subset **have** (j|‘(dom j − read-tmps (sb@sb′)) |‘ D) = j |‘ D
apply −
apply (rule ext)
apply (auto simp add: restrict-map-def)
done
moreover from D-subset D-tmps **have** D ⊆ dom (j |‘ (dom j − read-tmps (sb@sb′)))
by simp
moreover
note valid-sop [OF sop this]
ultimately have f-v′: f (j|‘(dom j − read-tmps (sb@sb′))) = v
by (simp add: f-v)

interpret causal′: causal-program-history is_{sb} sb@sb′ **by** fact

from is
have Write volatile a sop A L R W# instrs (sb @ sb′) @ is_{sb} = is @ prog-instrs (sb @ sb′)
by (simp add: $\text{Write}_{\mathsf{sb}}$)
with causal′.causal-program-history [of [], simplified, OF refl]
obtain is′ **where** is: is=Write volatile a sop A L R W#is′ **and**
is′: instrs (sb @ sb′) @ is_{sb} = is′ @ prog-instrs (sb @ sb′)
by auto

from ts-i is
have ts-i: ts!i = (p,Write volatile a sop A L R W#is′,
j |‘ (dom j − read-tmps (sb@sb′)),x,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by (simp add: $\text{Write}_{\mathsf{sb}}$)

from p **have** p′: p = hd-prog p_{sb} (sb@sb′)
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ hd-prog-idem)

from p_{sb} **have** p_{sb}′: last-prog p_{sb} (sb @ sb′) = p_{sb}
by (simp add: $\text{Write}_{\mathsf{sb}}$)

show ?thesis
proof (cases volatile)
case False
have memop-step:
(Write volatile a sop A L R W#is′,j|‘(dom j − read-tmps (sb@sb′)),
x,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) →
(is′,j|‘ (dom j − read-tmps (sb@sb′)),x,m(a:=v),$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$)
using D-subset
apply (simp only: sop f-v′ [symmetric] False)
apply (rule direct-memop-step.WriteNonVolatile)
done

let ?ts′ = ts[i := (p, is′,j |‘ (dom j − read-tmps (sb @ sb′)),x, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)]
from direct-computation.Memop [OF i-bound ts-i memop-step]
have (ts, m, $\mathcal{S}$) $\Rightarrow_{\mathsf{d}}$ (?ts′, m(a := v), $\mathcal{S}$).

also
from reads-consis **have** reads-consis′: reads-consistent True $\mathcal{O}'$ (m(a:=v)) sb
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ False)
from i-bound **have** i-bound′: i < length ?ts′
by auto

from i-bound
have ts′-i: ?ts′ ! i = (p, is′,j |‘ (dom j − read-tmps (sb @ sb′)), x, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)
by simp

from Cons.hyps [OF i-bound′ is′ causal′ ts′-i p′ p_{sb}′ reads-consis′ hist-consis′ valid-sops′ dist′ vol-read-consis′] i-bound
obtain is″ **where**
is″: instrs sb′ @ is_{sb} = is″ @ prog-instrs sb′ **and**
steps: (?ts′,m(a:=v),$\mathcal{S}$) $\Rightarrow_{\mathsf{d}}^*$
(ts[i := (last-prog (hd-prog p_{sb} sb′) sb, is″,
j |‘ (dom j − read-tmps sb′), x,
$\mathcal{D}$ ∨ outstanding-refs is-volatile-$\text{Write}_{\mathsf{sb}}$ sb ≠ {}, acquired True sb $\mathcal{O}$, release sb (dom $\mathcal{S}$) $\mathcal{R}$)],
flush sb (m(a := v)),share sb $\mathcal{S}$)
by (auto simp del: fun-upd-apply)
note steps
finally
show ?thesis
using is″
by (simp add: $\text{Write}_{\mathsf{sb}}$ False)
next
case True
have memop-step:
(Write volatile a sop A L R W#is′,j|‘(dom j − read-tmps (sb@sb′)),
x,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) →
(is′,j|‘ (dom j − read-tmps (sb@sb′)),x,m(a:=v),True,$\mathcal{O}$ ∪ A − R,Map.empty,$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
using D-subset

apply (simp only: sop f-v′ [symmetric] True)
apply (rule direct-memop-step.WriteVolatile)
done

let ?ts′ = ts[i := (p, is′, j |' (dom j − read-tmps (sb @ sb′)),x, True, $\mathcal{O}$ ∪ A − R,Map.empty)]
from direct-computation.Memop [OF i-bound ts-i memop-step]
have (ts, m, $\mathcal{S}$) $\Rightarrow_{\mathsf{d}}$ (?ts′, m(a := v), $\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L).

also
from reads-consis **have** reads-consis′: reads-consistent True ($\mathcal{O}$′ ∪ A − R)(m(a:=v)) sb
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ True)
from i-bound **have** i-bound′: i < length ?ts′
by auto

from i-bound
have ts′-i: ?ts′ ! i = (p, is′,j |' (dom j − read-tmps (sb @ sb′)), x, True, $\mathcal{O}$ ∪ A − R,Map.empty)
by simp

from Cons.hyps [OF i-bound′ is′ causal′ ts′-i p′ p_{sb}′ reads-consis′ hist-consis′ valid-sops′ dist′ vol-read-consis′, of ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)] i-bound
obtain is″ **where**
is″: instrs sb′ @ is_{sb} = is″ @ prog-instrs sb′ **and**
steps: (?ts′,m(a:=v),$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $\Rightarrow_{\mathsf{d}}^*$
(ts[i := (last-prog (hd-prog p_{sb} sb′) sb, is″,
j |' (dom j − read-tmps sb′), x,
True, acquired True sb ($\mathcal{O}$ ∪ A − R),release sb (dom ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) Map.empty)],
flush sb (m(a := v)), share sb ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L))
by (auto simp del: fun-upd-apply)
note steps
finally
show ?thesis
using is″
by (simp add: $\text{Write}_{\mathsf{sb}}$ True)
qed
next
case ($\text{Read}_{\mathsf{sb}}$ volatile a t v)

from vol-read-consis reads-consis **obtain** v: v=m a **and** r-consis: reads-consistent True $\mathcal{O}$′ m sb **and**
vol-read-consis′: volatile-reads-consistent m sb
by (cases volatile) (auto simp add: $\text{Read}_{\mathsf{sb}}$)

from valid-sops **have** valid-sops′: ∀sop ∈ write-sops sb. valid-sop sop
by (simp add: $\text{Read}_{\mathsf{sb}}$)

from hist-consis **obtain** j: j t = Some v **and**
hist-consis′: history-consistent j p (sb@sb′)

by (simp add: $\mathrm{Read}_{\mathsf{sb}}$ split: option.splits)
from dist **obtain** t-notin: t $\notin$ read-tmps (sb@sb′) **and**
dist′: distinct-read-tmps (sb@sb′) **by** (simp add: $\mathrm{Read}_{\mathsf{sb}}$)
from j t-notin **have** restrict-commute:
(j|‘ (dom j − read-tmps (sb@sb′)))(t↦v) =
j|‘ (dom j − read-tmps (sb@sb′))
apply −
apply (rule ext)
apply (auto simp add: restrict-map-def domIff)
done
from j t-notin
have restrict-commute′:
((j |‘ (dom j − insert t (read-tmps (sb@sb′))))(t ↦ v)) =
j|‘ (dom j − read-tmps (sb@sb′))
apply −
apply (rule ext)
apply (auto simp add: restrict-map-def domIff)
done

interpret causal′: causal-program-history $\mathrm{is}_{\mathsf{sb}}$ sb@sb′ **by** fact

from is
have Read volatile a t # instrs (sb @ sb′) @ $\mathrm{is}_{\mathsf{sb}}$ = is @ prog-instrs (sb @ sb′)
by (simp add: $\mathrm{Read}_{\mathsf{sb}}$)

with causal′.causal-program-history [of [], simplified, OF refl]
obtain is′ **where** is: is=Read volatile a t#is′ **and**
is′: instrs (sb @ sb′) @ $\mathrm{is}_{\mathsf{sb}}$ = is′ @ prog-instrs (sb @ sb′)
by auto

from ts-i is
have ts-i: ts!i = (p,Read volatile a t#is′,
j |‘ (dom j − insert t (read-tmps (sb@sb′))),x,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by (simp add: $\mathrm{Read}_{\mathsf{sb}}$)

from direct-memop-step.Read [of volatile a t is′ j|‘ (dom j − insert t (read-tmps (sb@sb′))) x m $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ $\mathcal{S}$]
have memop-step:
(Read volatile a t # is′,
j |‘ (dom j − insert t (read-tmps (sb @ sb′))), x, m, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) →
(is′,
j |‘ (dom j − (read-tmps (sb @ sb′))), x, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$,$\mathcal{S}$)
by (simp add: v [symmetric] restrict-commute restrict-commute′)

let ?ts′ = ts[i := (p, is′,
j |‘ (dom j − read-tmps (sb @ sb′)),x, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)]

from direct-computation.Memop [OF i-bound ts-i memop-step]
have (ts, m, $\mathcal{S}$) $\Rightarrow_{\mathsf{d}}$ (?ts′, m, $\mathcal{S}$).

also

from i-bound **have** i-bound′: i < length ?ts′
by auto

from i-bound
have ts′-i: ?ts′!i = (p,is′, (j |‘ (dom j − read-tmps (sb @ sb′))),x,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$)
by auto

from p **have** p′: p = hd-prog p_{sb} (sb@sb′)
by (auto simp add: Read_{sb} hd-prog-idem)

from p_{sb} **have** p_{sb}′: last-prog p_{sb} (sb @ sb′) = p_{sb}
by (simp add: Read_{sb})

from Cons.hyps [OF i-bound′ is′ causal′ ts′-i p′ p_{sb}′ r-consis hist-consis′
valid-sops′ dist′ vol-read-consis′]

obtain is″ **where**
is″: instrs sb′ @ is_{sb} = is″ @ prog-instrs sb′ **and**
steps: (?ts′,m,$\mathcal{S}$) $\Rightarrow_{d}{}^{*}$
(ts[i := (last-prog (hd-prog p_{sb} sb′) sb, is″,
j |‘ (dom j − read-tmps sb′),x, $\mathcal{D}$ ∨ outstanding-refs is-volatile-Write_{sb} sb ≠ {},
acquired True sb $\mathcal{O}$, release sb (dom $\mathcal{S}$) $\mathcal{R}$)],
flush sb m,share sb $\mathcal{S}$)
by (auto simp del: fun-upd-apply)

note steps
finally
show ?thesis
using is″
by (simp add: Read_{sb})
next
case (Ghost_{sb} A L R W)

from vol-read-consis
have vol-read-consis′: volatile-reads-consistent m sb
by (auto simp add: Ghost_{sb})

from reads-consis **have** r-consis: reads-consistent True ($\mathcal{O}$′ ∪ A − R) m sb
by (auto simp add: Ghost_{sb})

from valid-sops **have** valid-sops′: ∀sop ∈ write-sops sb. valid-sop sop
by (simp add: Ghost_{sb})

from hist-consis **obtain**
hist-consis′: history-consistent j p (sb@sb′)
by (simp add: Ghost_{sb})

from dist **obtain**
dist′: distinct-read-tmps (sb@sb′) **by** (simp add: $Ghost_{sb}$)

interpret causal′: causal-program-history is_{sb} sb@sb′ **by** fact

from is
have Ghost A L R W# instrs (sb @ sb′) @ is_{sb} = is @ prog-instrs (sb @ sb′)
by (simp add: $Ghost_{sb}$)

with causal′.causal-program-history [of [], simplified, OF refl]
obtain is′ **where** is: is=Ghost A L R W#is′ **and**
is′: instrs (sb @ sb′) @ is_{sb} = is′ @ prog-instrs (sb @ sb′)
by auto

from ts-i is
have ts-i: ts!i = (p,Ghost A L R W#is′,
j |‘ (dom j − (read-tmps (sb@sb′))),x,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by (simp add: $Ghost_{sb}$)

from direct-memop-step.Ghost [of A L R W is′
j|‘ (dom j − (read-tmps (sb@sb′))) x m $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ $\mathcal{S}$]
have memop-step:
(Ghost A L R W# is′,j |‘ (dom j − read-tmps (sb @ sb′)), x, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$)
→ (is′,j |‘ (dom j − read-tmps (sb @ sb′)), x, m, $\mathcal{D}$, $\mathcal{O}$ ∪ A − R , augment-rels (dom $\mathcal{S}$) R $\mathcal{R}$,
$\mathcal{S}$ $\oplus_{W}$ R $\ominus_{A}$ L)**.**

let ?ts′ = ts[i := (p, is′,
j |‘ (dom j − read-tmps (sb @ sb′)),x, $\mathcal{D}$, $\mathcal{O}$ ∪ A − R, augment-rels (dom $\mathcal{S}$) R $\mathcal{R}$)]
from direct-computation.Memop [OF i-bound ts-i memop-step]
have (ts, m, $\mathcal{S}$) $\Rightarrow_{d}$ (?ts′, m, $\mathcal{S}$ $\oplus_{W}$ R $\ominus_{A}$ L)**.**

also

from i-bound **have** i-bound′: i < length ?ts′
by auto

from i-bound
have ts′-i: ?ts′!i = (p,is′,(j |‘ (dom j − read-tmps (sb @ sb′))),x, $\mathcal{D}$, $\mathcal{O}$ ∪ A − R,augment-rels (dom $\mathcal{S}$) R $\mathcal{R}$)
by auto

from p **have** p′: p = hd-prog p_{sb} (sb@sb′)
by (auto simp add: $Ghost_{sb}$ hd-prog-idem)

from p_{sb} **have** p_{sb}′: last-prog p_{sb} (sb @ sb′) = p_{sb}
by (simp add: $Ghost_{sb}$)

from Cons.hyps [OF i-bound′ is′ causal′ ts′-i p′ p_{sb}′ r-consis hist-consis′

valid-sops′ dist′ vol-read-consis′, of $\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L]
obtain is′′ **where**
is′′: instrs sb′ @ is_{sb} = is′′ @ prog-instrs sb′ **and**
steps: (?ts′,m,$\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $\Rightarrow_{\mathsf{d}}^*$
(ts[i := (last-prog (hd-prog p_{sb} sb′) sb, is′′,
j |‘ (dom j − read-tmps sb′),x,
$\mathcal{D}$ ∨ outstanding-refs is-volatile-Write_{sb} sb ≠ {}, acquired True sb ($\mathcal{O}$ ∪ A − R),
release sb (dom ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) (augment-rels (dom $\mathcal{S}$) R $\mathcal{R}$))],
flush sb m,share sb ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L))
by (auto simp add: list-update-overwrite simp del: fun-upd-apply)

note steps
finally
show ?thesis
using is′′
by (simp add: Ghost_{sb})
qed
qed

corollary (**in** program) flush-store-buffer:
assumes i-bound: i < length ts
assumes instrs: instrs sb @ is_{sb} = is @ prog-instrs sb
assumes cph: causal-program-history is_{sb} sb
assumes ts-i: ts!i = (p,is,j |‘ (dom j − read-tmps sb),x,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
assumes p: p=hd-prog p_{sb} sb
assumes last-prog: (last-prog p_{sb} sb) = p_{sb}
assumes reads-consis: reads-consistent True $\mathcal{O}'$ m sb
assumes hist-consis: history-consistent j p sb
assumes valid-sops: ∀ sop ∈ write-sops sb. valid-sop sop
assumes dist: distinct-read-tmps sb
assumes vol-read-consis: volatile-reads-consistent m sb
shows (ts,m,$\mathcal{S}$) $\Rightarrow_{\mathsf{d}}^*$
(ts[i:=(p_{sb},is_{sb}, j,x,
$\mathcal{D}$ ∨ outstanding-refs is-volatile-Write_{sb} sb ≠ {},acquired True sb $\mathcal{O}$, release sb (dom $\mathcal{S}$) $\mathcal{R}$)],
flush sb m,share sb $\mathcal{S}$)
using flush-store-buffer-append [**where** sb′=[], simplified, OF i-bound instrs cph ts-i [simplified] p last-prog reads-consis hist-consis valid-sops dist vol-read-consis] last-prog
by simp

lemma last-prog-same-append: $\bigwedge$xs p_{sb}. last-prog p_{sb} (sb@xs) = p_{sb} ⟹ last-prog p_{sb} xs = p_{sb}
apply (induct sb)
apply simp
subgoal for a sb xs p_{sb}
apply (case-tac a)
apply simp
apply simp
apply simp

apply (drule last-prog-to-last-prog-same)
apply simp
apply simp
done
done

lemma reads-consistent-drop-volatile-writes-no-volatile-reads:
$\bigwedge$pending-write $\mathcal{O}$ m. reads-consistent pending-write $\mathcal{O}$ m sb $\Longrightarrow$
outstanding-refs is-volatile-Read$_{sb}$ ((dropWhile (Not $\circ$ is-volatile-Write$_{sb}$)) sb) = {}
apply (induct sb)
apply (auto split: memref.splits)
done

lemma reads-consistent-flush-other:
assumes no-volatile-Write$_{sb}$-sb: outstanding-refs is-volatile-Write$_{sb}$ sb = {}
shows $\bigwedge$m pending-write $\mathcal{O}$.
[[outstanding-refs (Not $\circ$ is-volatile-Read$_{sb}$) xs $\cap$ outstanding-refs is-non-volatile-Write$_{sb}$ sb = {};
reads-consistent pending-write $\mathcal{O}$ m xs]] $\Longrightarrow$ reads-consistent pending-write $\mathcal{O}$ (flush sb m) xs
proof (induct xs)
case Nil **thus** ?case **by** simp
next
case (Cons x xs)
note no-inter = ‹outstanding-refs (Not $\circ$ is-volatile-Read$_{sb}$) (x # xs) $\cap$
outstanding-refs is-non-volatile-Write$_{sb}$ sb = {}›
hence no-inter′: outstanding-refs (Not $\circ$ is-volatile-Read$_{sb}$) xs $\cap$ outstanding-refs is-non-volatile-Write$_{sb}$ sb = {}
by (auto)
note consis = ‹reads-consistent pending-write $\mathcal{O}$ m (x # xs)›
show ?case
proof (cases x)
case (Write$_{sb}$ volatile a sop v A L R)
show ?thesis
proof (cases volatile)
case False
from consis **obtain** consis′: reads-consistent pending-write $\mathcal{O}$ (m(a := v)) xs
by (simp add: Write$_{sb}$ False)
from Cons.hyps [OF no-inter′ consis′]
have reads-consistent pending-write $\mathcal{O}$ (flush sb (m(a := v))) xs**.**
moreover
from no-inter **have** a $\notin$ outstanding-refs is-non-volatile-Write$_{sb}$ sb
by (auto simp add: Write$_{sb}$ split: if-split-asm)

from flush-update-other′ [OF this no-volatile-Write$_{sb}$-sb]
have (flush sb (m(a := v))) = (flush sb m)(a := v)**.**

ultimately
show ?thesis
by (simp add: $\text{Write}_{\mathsf{sb}}$ False)
next
case True
from consis **obtain** consis′: reads-consistent True ($\mathcal{O} \cup A - R$) (m(a := v)) xs **and**
no-read: (outstanding-refs is-volatile-$\text{Read}_{\mathsf{sb}}$ xs = {})
by (simp add: $\text{Write}_{\mathsf{sb}}$ True)
from Cons.hyps [OF no-inter′ consis′]
have reads-consistent True ($\mathcal{O} \cup A - R$) (flush sb (m(a := v))) xs.
moreover
from no-inter **have** a $\notin$ outstanding-refs is-non-volatile-$\text{Write}_{\mathsf{sb}}$ sb
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ split: if-split-asm)

from flush-update-other′ [OF this no-volatile-$\text{Write}_{\mathsf{sb}}$-sb]
have (flush sb (m(a := v))) = (flush sb m)(a := v).
ultimately
show ?thesis
using no-read
by (simp add: $\text{Write}_{\mathsf{sb}}$ True)
qed
next
case ($\text{Read}_{\mathsf{sb}}$ volatile a t v)
from consis **obtain** val: ($\neg$ volatile $\longrightarrow$ (pending-write $\vee$ a $\in \mathcal{O}$) $\longrightarrow$ v = m a) **and**
consis′: reads-consistent pending-write $\mathcal{O}$ m xs
by (simp add: $\text{Read}_{\mathsf{sb}}$)
from Cons.hyps [OF no-inter′ consis′]
have hyp: reads-consistent pending-write $\mathcal{O}$ (flush sb m) xs
by simp
show ?thesis
proof (cases volatile)
case False
from no-inter False **have** a $\notin$ outstanding-refs is-non-volatile-$\text{Write}_{\mathsf{sb}}$ sb
by (auto simp add: $\text{Read}_{\mathsf{sb}}$ split: if-split-asm)
with no-volatile-$\text{Write}_{\mathsf{sb}}$-sb
have a $\notin$ outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ sb
apply (clarsimp simp add: outstanding-refs-conv is-$\text{Write}_{\mathsf{sb}}$-def split: memref.splits)
apply force
done
with hyp val flush-unchanged-addresses [OF this]
show ?thesis
by (simp add: $\text{Read}_{\mathsf{sb}}$)
next
case True
with hyp val **show** ?thesis
by (simp add: $\text{Read}_{\mathsf{sb}}$)
qed
next
case $\text{Prog}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** auto
next

case $\text{Ghost}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** auto
qed
qed

lemma reads-consistent-flush-independent:
assumes no-volatile-$\text{Write}_{\mathsf{sb}}$-sb: outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ sb $\cap$ outstanding-refs is-non-volatile-$\text{Read}_{\mathsf{sb}}$ xs = {}
assumes consis: reads-consistent pending-write $\mathcal{O}$ m xs
shows reads-consistent pending-write $\mathcal{O}$ (flush sb m) xs
proof −
from flush-unchanged-addresses [**where** sb=sb **and** m=m] no-volatile-$\text{Write}_{\mathsf{sb}}$-sb
have $\forall$ a $\in$ outstanding-refs is-non-volatile-$\text{Read}_{\mathsf{sb}}$ xs. flush sb m a = m a
by auto
from reads-consistent-mem-eq-on-non-volatile-reads [OF this subset-refl consis]
show ?thesis **.**
qed

lemma reads-consistent-flush-all-until-volatile-write-aux:
assumes no-reads: outstanding-refs is-volatile-$\text{Read}_{\mathsf{sb}}$ xs = {}
shows $\bigwedge$m pending-write $\mathcal{O}'$. ⟦reads-consistent pending-write $\mathcal{O}'$ m xs; $\forall$ i < length ts.
let (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) = ts!i in
outstanding-refs (Not $\circ$ is-volatile-$\text{Read}_{\mathsf{sb}}$) xs $\cap$
outstanding-refs is-non-volatile-$\text{Write}_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) = {}⟧
$\Longrightarrow$ reads-consistent pending-write $\mathcal{O}'$ (flush-all-until-volatile-write ts m) xs
proof (induct ts)
case Nil **thus** ?case **by** simp
next
case (Cons t ts)
have consis: reads-consistent pending-write $\mathcal{O}'$ m xs **by** fact

obtain p_{t} is_{t} $\mathcal{O}_{\mathsf{t}}$ $\mathcal{R}_{\mathsf{t}}$ $\mathcal{D}_{\mathsf{t}}$ j_{t} sb_{t}
where t: t=(p_{t},is_{t},j_{t},sb_{t},$\mathcal{D}_{\mathsf{t}}$,$\mathcal{O}_{\mathsf{t}}$,$\mathcal{R}_{\mathsf{t}}$)
by (cases t)

from Cons.prems t **obtain**
no-inter: outstanding-refs (Not $\circ$ is-volatile-$\text{Read}_{\mathsf{sb}}$) xs $\cap$
outstanding-refs is-non-volatile-$\text{Write}_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{t}) = {} **and**
no-inter′: $\forall$ i < length ts.
let (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) = ts!i in
outstanding-refs (Not $\circ$ is-volatile-$\text{Read}_{\mathsf{sb}}$) xs $\cap$
outstanding-refs is-non-volatile-$\text{Write}_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) = {}
by (force simp add: Let-def simp del: o-apply)

have out1: outstanding-refs is-volatile-$\text{Write}_{\mathsf{sb}}$

(takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_t$) = {}
by (auto simp add: outstanding-refs-conv dest: set-takeWhileD)

from no-inter **have** outstanding-refs (Not ∘ is-volatile-Read$_{sb}$) xs ∩
outstanding-refs is-non-volatile-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_t$) =
{}
by auto

from reads-consistent-flush-other [OF out1 this consis]

have reads-consistent pending-write $\mathcal{O}'$ (flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_t$)
m) xs.
from Cons.hyps [OF this no-inter′]
show ?case
by (simp add: t)
qed

lemma reads-consistent-flush-other′:
assumes no-volatile-Write$_{sb}$-sb: outstanding-refs is-volatile-Write$_{sb}$ sb = {}
shows ⋀m $\mathcal{O}$.
⟦outstanding-refs is-non-volatile-Write$_{sb}$ sb ∩
(outstanding-refs is-volatile-Write$_{sb}$ xs ∪
outstanding-refs is-non-volatile-Write$_{sb}$ xs ∪
outstanding-refs is-non-volatile-Read$_{sb}$ (dropWhile (Not ∘ is-volatile-Write$_{sb}$) xs)
∪
(outstanding-refs is-non-volatile-Read$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) xs)
− RO) ∪
($\mathcal{O}$ ∪ all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) xs))
) = {};
reads-consistent False $\mathcal{O}$ m xs;
read-only-reads $\mathcal{O}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) xs) ⊆ RO⟧
⟹ reads-consistent False $\mathcal{O}$ (flush sb m) xs
proof (induct xs)
case Nil **thus** ?case **by** simp
next
case (Cons x xs)

note no-inter = Cons.prems (1)

note consis = ‹reads-consistent False $\mathcal{O}$ m (x # xs)›
have aargh: (Not ∘ is-volatile-Write$_{sb}$) = (λa. ¬ is-volatile-Write$_{sb}$ a)
by (rule ext) auto

note RO = ‹read-only-reads $\mathcal{O}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) (x#xs)) ⊆ RO›

show ?case

proof (cases x)
case (Write$_{sb}$ volatile a sop v A L R)
show ?thesis
proof (cases volatile)
case False
from consis **obtain** consis′: reads-consistent False $\mathcal{O}$ (m(a := v)) xs
by (simp add: Write$_{sb}$ False)

from no-inter
have no-inter′: outstanding-refs is-non-volatile-Write$_{sb}$ sb ∩
(outstanding-refs is-volatile-Write$_{sb}$ xs ∪
outstanding-refs is-non-volatile-Write$_{sb}$ xs ∪
outstanding-refs is-non-volatile-Read$_{sb}$ (dropWhile (Not ∘ is-volatile-Write$_{sb}$) xs) ∪
(outstanding-refs is-non-volatile-Read$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) xs) − RO) ∪
($\mathcal{O}$ ∪ all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) xs))
) = {}
by (clarsimp simp add: Write$_{sb}$ False aargh)

from RO
have RO′: read-only-reads $\mathcal{O}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) xs) ⊆ RO
by (auto simp add: Write$_{sb}$ False)

from Cons.hyps [OF no-inter′ consis′ RO′]
have reads-consistent False $\mathcal{O}$ (flush sb (m(a := v))) xs**.**
moreover
from no-inter **have** a ∉ outstanding-refs is-non-volatile-Write$_{sb}$ sb
by (auto simp add: Write$_{sb}$ split: if-split-asm)

from flush-update-other′ [OF this no-volatile-Write$_{sb}$-sb]
have (flush sb (m(a := v))) = (flush sb m)(a := v)**.**
ultimately
show ?thesis
by (simp add: Write$_{sb}$ False)
next
case True
from consis **obtain** consis′: reads-consistent True ($\mathcal{O}$ ∪ A − R) (m(a := v)) xs **and**
no-read: (outstanding-refs is-volatile-Read$_{sb}$ xs = {})
by (simp add: Write$_{sb}$ True)

from no-inter **obtain**
a-notin: a ∉ outstanding-refs is-non-volatile-Write$_{sb}$ sb **and**
disj: (outstanding-refs (Not ∘ is-volatile-Read$_{sb}$) xs) ∩
outstanding-refs is-non-volatile-Write$_{sb}$ sb = {}
by (auto simp add: Write$_{sb}$ True aargh misc-outstanding-refs-convs)

from reads-consistent-flush-other [OF no-volatile-Write$_{sb}$-sb disj consis′]

have reads-consistent True ($\mathcal{O} \cup A - R$) (flush sb (m(a := v))) xs.
moreover
note a-notin

from flush-update-other′ [OF this no-volatile-Write$_{sb}$-sb]
have (flush sb (m(a := v))) = (flush sb m)(a := v).
ultimately
show ?thesis
using no-read
by (simp add: Write$_{sb}$ True)
qed
next
case (Read$_{sb}$ volatile a t v)
from consis **obtain** val: ($\neg$ volatile $\longrightarrow$ a $\in \mathcal{O} \longrightarrow$ v = m a) **and**
consis′: reads-consistent False $\mathcal{O}$ m xs
by (simp add: Read$_{sb}$)

from RO
have RO′: read-only-reads $\mathcal{O}$ (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) xs) $\subseteq$ RO
by (auto simp add: Read$_{sb}$)

from no-inter
have no-inter′: outstanding-refs is-non-volatile-Write$_{sb}$ sb $\cap$
(outstanding-refs is-volatile-Write$_{sb}$ xs $\cup$
outstanding-refs is-non-volatile-Write$_{sb}$ xs $\cup$
outstanding-refs is-non-volatile-Read$_{sb}$ (dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) xs) $\cup$
(outstanding-refs is-non-volatile-Read$_{sb}$ (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) xs) $-$
RO) $\cup$
($\mathcal{O} \cup$ all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) xs))
) = {}
by (fastforce simp add: Read$_{sb}$ aargh)

show ?thesis
proof (cases volatile)
case True

from Cons.hyps [OF no-inter′ consis′ RO′]
show ?thesis
by (simp add: Read$_{sb}$ True)
next
case False
note non-volatile=this

from Cons.hyps [OF no-inter′ consis′ RO′]
have hyp: reads-consistent False $\mathcal{O}$ (flush sb m) xs.

show ?thesis
proof (cases a $\in \mathcal{O}$)

case False
with hyp **show** ?thesis
  by (simp add: $\text{Read}_{\mathsf{sb}}$ non-volatile False)
    next
case True
from no-inter True **have** a-notin: a $\notin$ outstanding-refs is-non-volatile-$\text{Write}_{\mathsf{sb}}$ sb
  by blast

with no-volatile-$\text{Write}_{\mathsf{sb}}$-sb
have a $\notin$ outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ sb
  apply (clarsimp simp add: outstanding-refs-conv is-$\text{Write}_{\mathsf{sb}}$-def split: memref.splits)
  apply force
  done

from flush-unchanged-addresses [OF this] hyp val

show ?thesis
  by (simp add: $\text{Read}_{\mathsf{sb}}$ non-volatile True)
    qed
   qed
 next
   case $\text{Prog}_{\mathsf{sb}}$ **with** Cons **show** ?thesis
     by auto
 next
   case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
   from consis **obtain** consis′: reads-consistent False ($\mathcal{O}$ ∪ A − R) m xs
     by (simp add: $\text{Ghost}_{\mathsf{sb}}$)

   from RO
   have RO′: read-only-reads ($\mathcal{O}$ ∪ A − R) (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) xs) ⊆
RO
     by (auto simp add: $\text{Ghost}_{\mathsf{sb}}$)

   from no-inter
   have no-inter′: outstanding-refs is-non-volatile-$\text{Write}_{\mathsf{sb}}$ sb ∩
     (outstanding-refs is-volatile-$\text{Write}_{\mathsf{sb}}$ xs ∪
       outstanding-refs is-non-volatile-$\text{Write}_{\mathsf{sb}}$ xs ∪
         outstanding-refs is-non-volatile-$\text{Read}_{\mathsf{sb}}$ (dropWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) xs)
∪
         (outstanding-refs is-non-volatile-$\text{Read}_{\mathsf{sb}}$ (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) xs)
− RO) ∪
        ($\mathcal{O}$ ∪ A − R ∪ all-acquired (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) xs))
) = {}
     by (fastforce simp add: $\text{Ghost}_{\mathsf{sb}}$ aargh)

   from Cons.hyps [OF no-inter′ consis′ RO′]
   show ?thesis
     by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
 qed

qed

lemma reads-consistent-flush-all-until-volatile-write-aux$'$:
assumes no-reads: outstanding-refs is-volatile-Read$_{sb}$ xs = {}
assumes read-only-reads-RO: read-only-reads $\mathcal{O}'$ (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) xs) $\subseteq$ RO
shows $\bigwedge$m. $[\![$reads-consistent False $\mathcal{O}'$ m xs; $\forall$i < length ts.
let (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$) = ts!i in
outstanding-refs is-non-volatile-Write$_{sb}$ (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\cap$
(outstanding-refs is-volatile-Write$_{sb}$ xs $\cup$
outstanding-refs is-non-volatile-Write$_{sb}$ xs $\cup$
outstanding-refs is-non-volatile-Read$_{sb}$ (dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) xs) $\cup$
(outstanding-refs is-non-volatile-Read$_{sb}$ (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) xs) $-$ RO) $\cup$
($\mathcal{O}'$ $\cup$ all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) xs))
)
= {}
$]\!]$
$\Longrightarrow$ reads-consistent False $\mathcal{O}'$ (flush-all-until-volatile-write ts m) xs
proof (induct ts)
case Nil **thus** ?case **by** simp
next
case (Cons t ts)
have consis: reads-consistent False $\mathcal{O}'$ m xs **by** fact

obtain p_t is_t $\mathcal{O}_t$ $\mathcal{R}_t$ $\mathcal{D}_t$ j_t sb_t
where t: t=(p_t,is_t,j_t,sb_t,$\mathcal{D}_t$,$\mathcal{O}_t$,$\mathcal{R}_t$)
by (cases t)

obtain
no-inter: outstanding-refs is-non-volatile-Write$_{sb}$ (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_t) $\cap$
(outstanding-refs is-volatile-Write$_{sb}$ xs $\cup$
outstanding-refs is-non-volatile-Write$_{sb}$ xs $\cup$
outstanding-refs is-non-volatile-Read$_{sb}$ (dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) xs) $\cup$
(outstanding-refs is-non-volatile-Read$_{sb}$ (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) xs) $-$ RO) $\cup$
($\mathcal{O}'$ $\cup$ all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) xs))

)
= {} **and**
no-inter$'$: $\forall$i < length ts.
let (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$) = ts!i in
outstanding-refs is-non-volatile-Write$_{sb}$ (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\cap$
(outstanding-refs is-volatile-Write$_{sb}$ xs $\cup$
outstanding-refs is-non-volatile-Write$_{sb}$ xs $\cup$

outstanding-refs is-non-volatile-Read$_{sb}$ (dropWhile (Not ∘ is-volatile-Write$_{sb}$) xs)
∪
(outstanding-refs is-non-volatile-Read$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) xs)
− RO) ∪
($\mathcal{O}'$ ∪ all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) xs))
)
= {}
proof −
show ?thesis
apply (rule that)
using Cons.prems (2) [rule-format, of 0]
apply (clarsimp simp add: t)
apply clarsimp
using Cons.prems (2)
apply −
subgoal for i
apply (drule-tac x=Suc i **in** spec)
apply (clarsimp simp add: Let-def simp del: o-apply)
done
done
qed

have out1: outstanding-refs is-volatile-Write$_{sb}$
(takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_t$) = {}
by (auto simp add: outstanding-refs-conv dest: set-takeWhileD)

from reads-consistent-flush-other′ [OF out1 no-inter consis read-only-reads-RO]
have reads-consistent False $\mathcal{O}'$ (flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_t$) m) xs**.**
from Cons.hyps [OF this no-inter′]
show ?case
by (simp add: t)
qed

lemma in-outstanding-refs-cases [consumes 1, case-names Write$_{sb}$ Read$_{sb}$]:
a ∈ outstanding-refs P xs ⟹
(⋀volatile sop v A L R W. (Write$_{sb}$ volatile a sop v A L R W) ∈ set xs ⟹ P
(Write$_{sb}$ volatile a sop v A L R W) ⟹ C) ⟹
(⋀volatile t v. (Read$_{sb}$ volatile a t v) ∈ set xs ⟹ P (Read$_{sb}$ volatile a t v) ⟹ C)
⟹ C
apply (clarsimp simp add: outstanding-refs-conv)
subgoal for x
apply (case-tac x)
apply fastforce+

done
done

lemma dropWhile-Cons: (dropWhile P xs) = x#ys $\Longrightarrow$ ¬ P x
apply (induct xs)
apply (auto split: if-split-asm)
done

lemma reads-consistent-dropWhile:
reads-consistent pending-write $\mathcal{O}$ m (dropWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb) =
reads-consistent True $\mathcal{O}$ m (dropWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb)
apply (case-tac (dropWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb))
apply (simp only:)
apply simp
apply (frule dropWhile-Cons)
apply (auto split: memref.splits)
done

theorem
reads-consistent-flush-all-until-volatile-write:
⋀i m pending-write. ⟦valid-ownership-and-sharing $\mathcal{S}$ ts;
i < length ts; ts!i = (p, is,j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$);
reads-consistent pending-write $\mathcal{O}$ m sb ⟧
$\Longrightarrow$ reads-consistent True (acquired True (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb) $\mathcal{O}$)
(flush-all-until-volatile-write ts m) (dropWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb)
proof (induct ts)
case Nil **thus** ?case **by** simp
next
case (Cons t ts)
note i-bound = ‹i < length (t # ts)›
note ts-i = ‹(t # ts) ! i = (p, is,j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)›
note consis = ‹reads-consistent pending-write $\mathcal{O}$ m sb›
note valid = ‹valid-ownership-and-sharing $\mathcal{S}$ (t#ts)›
then interpret valid-ownership-and-sharing $\mathcal{S}$ t#ts**.**
from valid-ownership-and-sharing-tl [OF valid] **have** valid′: valid-ownership-and-sharing $\mathcal{S}$ ts**.**

obtain p_{t} is_{t} $\mathcal{O}_{\mathsf{t}}$ $\mathcal{R}_{\mathsf{t}}$ $\mathcal{D}_{\mathsf{t}}$ j_{t} sb_{t}
where t: t=(p_{t},is_{t},j_{t},sb_{t},$\mathcal{D}_{\mathsf{t}}$,$\mathcal{O}_{\mathsf{t}}$,$\mathcal{R}_{\mathsf{t}}$)
by (cases t)
show ?case
proof (cases i)
case 0
with ts-i t **have** sb-eq: sb=sb_{t}
by simp

let ?take-sb = (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb)
let ?drop-sb = (dropWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb)

from reads-consistent-append [of pending-write $\mathcal{O}$ m ?take-sb ?drop-sb] consis
have consis$'$: reads-consistent True (acquired True ?take-sb $\mathcal{O}$) (flush ?take-sb m) ?drop-sb
apply (cases outstanding-refs is-volatile-Write$_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) $\neq$ {})
apply clarsimp
apply clarsimp
apply (simp add: reads-consistent-dropWhile [of pending-write])
done

from reads-consistent-drop-volatile-writes-no-volatile-reads [OF consis]
have no-vol-Read$_{\mathsf{sb}}$: outstanding-refs is-volatile-Read$_{\mathsf{sb}}$ (dropWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) = {}.
hence outstanding-refs (Not $\circ$ is-volatile-Read$_{\mathsf{sb}}$) (dropWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb)
=
outstanding-refs (λs. True) (dropWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb)
by (auto simp add: outstanding-refs-conv)

have $\forall$ i<length ts.
let (p, is,j, sb$'$, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) = ts ! i
in outstanding-refs (Not $\circ$ is-volatile-Read$_{\mathsf{sb}}$) (dropWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) $\cap$
outstanding-refs is-non-volatile-Write$_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$'$) = {}
proof –
{
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j x
assume j-bound: j < length ts
assume ts-j: ts!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
assume x-in-sb: x $\in$ outstanding-refs (Not $\circ$ is-volatile-Read$_{\mathsf{sb}}$) (dropWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb)
assume x-in-j: x $\in$ outstanding-refs is-non-volatile-Write$_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb_j)
have False
proof –
from outstanding-non-volatile-write-not-volatile-read-disj [rule-format, of Suc j 0, simplified, OF j-bound ts-j t]
sb-eq x-in-sb x-in-j
show ?thesis
by auto
qed
}
thus ?thesis
by (auto simp add: Let-def)
qed
from reads-consistent-flush-all-until-volatile-write-aux [OF no-vol-Read$_{\mathsf{sb}}$ consis$'$ this]
show ?thesis
by (simp add: t sb-eq del: o-apply)

next
case (Suc k)
with i-bound **have** k-bound: k < length ts
by auto

from ts-i Suc **have** ts-k: ts ! k = (p, is,j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)
by simp

have reads-consistent False $\mathcal{O}$ (flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_t$) m) sb
proof –
have no-vW:
outstanding-refs is-volatile-Write$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_t$) = {}
apply (clarsimp simp add: outstanding-refs-conv)
apply (drule set-takeWhileD)
apply simp
done

from consis **have** consis′: reads-consistent False $\mathcal{O}$ m sb
by (cases pending-write) (auto intro: reads-consistent-pending-write-antimono)
note disj = outstanding-non-volatile-write-disj [**where** i=0, OF - i-bound [simplified Suc], simplified, OF t ts-k]

from reads-consistent-flush-other′ [OF no-vW disj consis′ subset-refl]
show ?thesis **.**
qed
from Cons.hyps [OF valid′ k-bound ts-k this]
show ?thesis
by (simp add: t)
qed
qed

lemma split-volatile-Write$_{sb}$-in-outstanding-refs:
a ∈ outstanding-refs is-volatile-Write$_{sb}$ xs ⟹ (∃ sop v ys zs A L R W. xs = ys@(Write$_{sb}$ True a sop v A L R W#zs))
proof (induct xs)
case Nil **thus** ?case **by** simp
next
case (Cons x xs)
have a-in: a ∈ outstanding-refs is-volatile-Write$_{sb}$ (x # xs) **by** fact
show ?case
proof (cases x)
case (Write$_{sb}$ volatile a′ sop v A L R W)
show ?thesis
proof (cases volatile)
case False

from a-in **have** a ∈ outstanding-refs is-volatile-Write$_{sb}$ xs
by (auto simp add: False Write$_{sb}$)
from Cons.hyps [OF this] **obtain** sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$ ys zs
where xs=ys@Write$_{sb}$ True a sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$#zs
by auto
hence x#xs = (x#ys)@Write$_{sb}$ True a sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$#zs
by auto
thus ?thesis
by blast
next
case True
note volatile = this
show ?thesis
proof (cases a$'$=a)
case False
with a-in **have** a ∈ outstanding-refs is-volatile-Write$_{sb}$ xs
by (auto simp add: volatile Write$_{sb}$)
from Cons.hyps [OF this] **obtain** sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$ ys zs
where xs=ys@Write$_{sb}$ True a sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$#zs
by auto
hence x#xs = (x#ys)@Write$_{sb}$ True a sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$#zs
by auto
thus ?thesis
by blast
next
case True
then have x#xs=[]@(Write$_{sb}$ True a sop v A L R W#xs)
by (simp add: Write$_{sb}$ volatile True)
thus ?thesis
by blast
qed
qed
next
case Read$_{sb}$
from a-in **have** a ∈ outstanding-refs is-volatile-Write$_{sb}$ xs
by (auto simp add: Read$_{sb}$)
from Cons.hyps [OF this] **obtain** sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$ ys zs
where xs=ys@Write$_{sb}$ True a sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$#zs
by auto
hence x#xs = (x#ys)@Write$_{sb}$ True a sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$#zs
by auto
thus ?thesis
by blast
next
case Prog$_{sb}$
from a-in **have** a ∈ outstanding-refs is-volatile-Write$_{sb}$ xs
by (auto simp add: Prog$_{sb}$)
from Cons.hyps [OF this] **obtain** sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$ ys zs
where xs=ys@Write$_{sb}$ True a sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$#zs
by auto

hence x#xs = (x#ys)@Write$_{\mathsf{sb}}$ True a sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$#zs
by auto
thus ?thesis
by blast
next
case Ghost$_{\mathsf{sb}}$
from a-in **have** a $\in$ outstanding-refs is-volatile-Write$_{\mathsf{sb}}$ xs
by (auto simp add: Ghost$_{\mathsf{sb}}$)
from Cons.hyps [OF this] **obtain** sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$ ys zs
where xs=ys@Write$_{\mathsf{sb}}$ True a sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$#zs
by auto
hence x#xs = (x#ys)@Write$_{\mathsf{sb}}$ True a sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$#zs
by auto
thus ?thesis
by blast
qed
qed

lemma sharing-consistent-mono-shared:
$\bigwedge\mathcal{S}\ \mathcal{S}'\ \mathcal{O}$.
dom $\mathcal{S}$ $\subseteq$ dom $\mathcal{S}'$ $\Longrightarrow$ sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb $\Longrightarrow$ sharing-consistent $\mathcal{S}'$ $\mathcal{O}$ sb
apply (induct sb)
apply simp
subgoal for a sb $\mathcal{S}$ $\mathcal{S}'$ $\mathcal{O}$
apply (case-tac a)
apply clarsimp
subgoal for volatile a D f v A L R W
apply (frule-tac A=$\mathcal{S}$ **and** B=$\mathcal{S}'$ **and** C=R **and** x=W **in** augment-mono-aux)
apply (frule-tac A=$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R **and** B=$\mathcal{S}'$ $\oplus_{\mathsf{W}}$ R **and** C=L **in** restrict-mono-aux)
apply blast
done
apply clarsimp
apply clarsimp
apply clarsimp
subgoal for A L R W
apply (frule-tac A=$\mathcal{S}$ **and** B=$\mathcal{S}'$ **and** C=R **and** x=W **in** augment-mono-aux)
apply (frule-tac A=$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R **and** B=$\mathcal{S}'$ $\oplus_{\mathsf{W}}$ R **and** C=L **in** restrict-mono-aux)
apply blast
done
done
done

lemma sharing-consistent-mono-owns:
$\bigwedge\mathcal{O}\ \mathcal{O}'\ \mathcal{S}$.
$\mathcal{O}$ $\subseteq$ $\mathcal{O}'$ $\Longrightarrow$ sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb $\Longrightarrow$ sharing-consistent $\mathcal{S}$ $\mathcal{O}'$ sb
apply (induct sb)
apply simp
subgoal for a sb $\mathcal{O}$ $\mathcal{O}'$ $\mathcal{S}$
apply (case-tac a)
apply clarsimp

subgoal for volatile a D f v A L R W
apply (frule-tac A=$\mathcal{O}$ **and** B=$\mathcal{O}'$ **and** C=A **in** union-mono-aux)
apply (frule-tac A=$\mathcal{O}$ ∪ A **and** B=$\mathcal{O}'$ ∪ A **and** C=R **in** set-minus-mono-aux)
apply fastforce
done
apply clarsimp
apply clarsimp
apply clarsimp
subgoal for A L R W
apply (frule-tac A=$\mathcal{O}$ **and** B=$\mathcal{O}'$ **and** C=A **in** union-mono-aux)
apply (frule-tac A=$\mathcal{O}$ ∪ A **and** B=$\mathcal{O}'$ ∪ A **and** C=R **in** set-minus-mono-aux)
apply fastforce
done
done
done

primrec all-shared :: 'a memref list ⇒ addr set
where
all-shared [] = {}
| all-shared (i#is) =
(case i of
$\text{Write}_{\mathsf{sb}}$ volatile - - - A L R W ⇒ (if volatile then R ∪ all-shared is else all-shared is)
| $\text{Ghost}_{\mathsf{sb}}$ A L R W ⇒ R ∪ all-shared is
| - ⇒ all-shared is)

lemma sharing-consistent-all-shared:
$\bigwedge\mathcal{S}\ \mathcal{O}$. sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb ⟹ all-shared sb ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$
apply (induct sb)
apply clarsimp
subgoal for a
apply (case-tac a)
apply (fastforce split: memref.splits if-split-asm)
apply clarsimp
apply clarsimp
apply fastforce
done
done

lemma sharing-consistent-share-all-shared:
$\bigwedge\mathcal{S}$. dom (share sb $\mathcal{S}$) ⊆ dom $\mathcal{S}$ ∪ all-shared sb
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a sop t A L R W)
show ?thesis

```
    proof (cases volatile)
      case True
      from Cons.hyps [of (𝒮 ⊕W R ⊖A L)]
      show ?thesis
        by (auto simp add: Writesb True)
    next
      case False with Cons Writesb show ?thesis by auto
    qed
  next
    case Readsb with Cons show ?thesis by auto
  next
    case Progsb with Cons show ?thesis by auto
  next
    case (Ghostsb A L R W)
    from Cons.hyps [of (𝒮 ⊕W R ⊖A L)]
    show ?thesis
      by (auto simp add: Ghostsb)
  qed
qed
```

```
primrec all-unshared :: 'a memref list ⇒ addr set
where
  all-unshared [] = {}
| all-unshared (i#is) =
    (case i of
      Writesb volatile - - -  A L R W ⇒ (if volatile then L ∪ all-unshared is else all-unshared
is)
      | Ghostsb A L R W ⇒ L ∪ all-unshared is
      | - ⇒ all-unshared is)
```

```
lemma all-unshared-append: all-unshared (xs @ ys) = all-unshared xs ∪ all-unshared ys
  apply (induct xs)
  apply  simp
  subgoal for a
  apply (case-tac a)
  apply auto
  done
  done
```

```
lemma freshly-shared-owned:
  ⋀𝒮 𝒪. sharing-consistent 𝒮 𝒪 sb ⟹ dom (share sb 𝒮) − dom 𝒮 ⊆ 𝒪
proof (induct sb)
  case Nil thus ?case by simp
next
  case (Cons x sb)
  show ?case
```

```
  proof (cases x)
    case (Write_sb volatile a sop v A L R W)
    show ?thesis
    proof (cases volatile)
      case False
      with Cons Write_sb show ?thesis by auto
    next
      case True
      from Cons.hyps [where 𝒮=(𝒮 ⊕_W R ⊖_A L) and 𝒪=(𝒪 ∪ A − R)] Cons.prems
      show ?thesis
 by (auto simp add: Write_sb True)
    qed
  next
    case Read_sb with Cons show ?thesis by auto
  next
    case Prog_sb with Cons show ?thesis by auto
  next
    case (Ghost_sb A L R W)
    with Cons.hyps [where 𝒮=(𝒮 ⊕_W R ⊖_A L) and 𝒪=(𝒪 ∪ A − R)] Cons.prems show
?thesis by auto
  qed
qed

lemma unshared-all-unshared:
  ⋀𝒮 𝒪. sharing-consistent 𝒮 𝒪 sb ⟹ dom 𝒮 − dom (share sb 𝒮) ⊆ all-unshared sb
proof (induct sb)
  case Nil thus ?case by simp
next
  case (Cons x sb)
  show ?case
  proof (cases x)
    case (Write_sb volatile a sop v A L R W)
    show ?thesis
    proof (cases volatile)
      case False
      with Cons Write_sb show ?thesis by auto
    next
      case True
      from Cons.hyps [where 𝒮=(𝒮 ⊕_W R ⊖_A L) and 𝒪=(𝒪 ∪ A − R)] Cons.prems
      show ?thesis
 by (auto simp add: Write_sb True)
    qed
  next
    case Read_sb with Cons show ?thesis by auto
  next
    case Prog_sb with Cons show ?thesis by auto
  next
    case (Ghost_sb A L R W)
    with Cons.hyps [where 𝒮=(𝒮 ⊕_W R ⊖_A L) and 𝒪=(𝒪 ∪ A − R)] Cons.prems show
?thesis by auto
```

qed
qed

lemma unshared-acquired-or-owned:
$\bigwedge\mathcal{S}\ \mathcal{O}$. sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb $\Longrightarrow$ all-unshared sb $\subseteq$ all-acquired sb $\cup$ $\mathcal{O}$
apply (induct sb)
apply simp
subgoal for a
apply (case-tac a)
apply auto+
done
done

lemma all-shared-acquired-or-owned:
$\bigwedge\mathcal{S}\ \mathcal{O}$. sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb $\Longrightarrow$ all-shared sb $\subseteq$ all-acquired sb $\cup$ $\mathcal{O}$
apply (induct sb)
apply simp
subgoal for a
apply (case-tac a)
apply auto+
done
done

lemma sharing-consistent-preservation:
$\bigwedge\mathcal{S}\ \mathcal{S}'\ \mathcal{O}$.
$[\![$sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb;
all-acquired sb $\cap$ dom $\mathcal{S}$ $-$ dom $\mathcal{S}'$ = {};
all-unshared sb $\cap$ dom $\mathcal{S}'$ $-$ dom $\mathcal{S}$ = {}$]\!]$
$\Longrightarrow$ sharing-consistent $\mathcal{S}'$ $\mathcal{O}$ sb
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
have consis: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ (x # sb) **by** fact
have removed-cond: all-acquired (x # sb) $\cap$ dom $\mathcal{S}$ $-$ dom $\mathcal{S}'$ = {} **by** fact
have new-cond: all-unshared (x # sb) $\cap$ dom $\mathcal{S}'$ $-$ dom $\mathcal{S}$ = {} **by** fact
show ?case
proof (cases x)
case (Write$_{\mathsf{sb}}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)
case False **with** Write$_{\mathsf{sb}}$ Cons **show** ?thesis
by auto
next
case True
from consis **obtain**
A: A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ **and**
L: L $\subseteq$ A **and**
A-R: A $\cap$ R = {} **and**

R: R $\subseteq \mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb
by (clarsimp simp add: Write$_{\mathsf{sb}}$ True)

from removed-cond **obtain** rem-cond: (A $\cup$ all-acquired sb) $\cap$ dom $\mathcal{S} \subseteq$ dom $\mathcal{S}'$ **by** (clarsimp simp add: Write$_{\mathsf{sb}}$ True)
hence rem-cond′: all-acquired sb $\cap$ dom ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $-$ dom ($\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = {}
by auto

from new-cond **obtain** (L $\cup$ all-unshared sb) $\cap$ dom $\mathcal{S}' \subseteq$ dom $\mathcal{S}$ **by** (clarsimp simp add: Write$_{\mathsf{sb}}$ True)
hence new-cond′: all-unshared sb $\cap$ dom ($\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $-$ dom ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = {}
by auto

from Cons.hyps [OF consis′ rem-cond′ new-cond′]
have sharing-consistent ($\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb**.**
moreover
from A rem-cond **have** A $\subseteq$ dom $\mathcal{S}' \cup \mathcal{O}$
by auto
moreover note L A-R R
ultimately show ?thesis
by (auto simp add: Write$_{\mathsf{sb}}$ True)
qed
next
case (Ghost$_{\mathsf{sb}}$ A L R W)
from consis **obtain**
A: A $\subseteq$ dom $\mathcal{S} \cup \mathcal{O}$ **and**
L: L $\subseteq$ A **and**
A-R: A $\cap$ R = {} **and**
R: R $\subseteq \mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb
by (clarsimp simp add: Ghost$_{\mathsf{sb}}$)

from removed-cond **obtain** rem-cond: (A $\cup$ all-acquired sb) $\cap$ dom $\mathcal{S} \subseteq$ dom $\mathcal{S}'$ **by** (clarsimp simp add: Ghost$_{\mathsf{sb}}$)
hence rem-cond′: all-acquired sb $\cap$ dom ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $-$ dom ($\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = {}
by auto

from new-cond **obtain** (L $\cup$ all-unshared sb) $\cap$ dom $\mathcal{S}' \subseteq$ dom $\mathcal{S}$ **by** (clarsimp simp add: Ghost$_{\mathsf{sb}}$)
hence new-cond′: all-unshared sb $\cap$ dom ($\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $-$ dom ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = {}
by auto

from Cons.hyps [OF consis′ rem-cond′ new-cond′]
have sharing-consistent ($\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb**.**

moreover
from A rem-cond **have** A $\subseteq$ dom $\mathcal{S}'$ $\cup$ $\mathcal{O}$
by auto
moreover note L A-R R
ultimately show ?thesis
by (auto simp add: $\mathrm{Ghost}_{\mathsf{sb}}$)
qed (insert Cons, auto)
qed

lemma (**in** sharing-consis) sharing-consis-preservation:
assumes dist:
$\forall$ i < length ts. let (-,-,-,sb,-,-,-) = ts!i in
all-acquired sb $\cap$ dom $\mathcal{S}$ $-$ dom $\mathcal{S}'$ = {} $\wedge$ all-unshared sb $\cap$ dom $\mathcal{S}'$ $-$ dom $\mathcal{S}$ =
{}
shows sharing-consis $\mathcal{S}'$ ts
proof
fix i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
assume i-bound: i < length ts
assume ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
show sharing-consistent $\mathcal{S}'$ $\mathcal{O}$ sb
proof $-$
from sharing-consis [OF i-bound ts-i]
have consis: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb**.**
from dist [rule-format, OF i-bound] ts-i
obtain
acq: all-acquired sb $\cap$ dom $\mathcal{S}$ $-$ dom $\mathcal{S}'$ = {} **and**
uns: all-unshared sb $\cap$ dom $\mathcal{S}'$ $-$ dom $\mathcal{S}$ = {}
by auto
from sharing-consistent-preservation [OF consis acq uns]
show ?thesis **.**
qed
qed

lemma (**in** sharing-consis) sharing-consis-shared-exchange:
assumes dist:
$\forall$ i < length ts. let (-,-,-,sb,-,-,-) = ts!i in
$\forall$ a $\in$ all-acquired sb. $\mathcal{S}'$ a = $\mathcal{S}$ a
shows sharing-consis $\mathcal{S}'$ ts
proof
fix i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
assume i-bound: i < length ts
assume ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
show sharing-consistent $\mathcal{S}'$ $\mathcal{O}$ sb
proof $-$
from sharing-consis [OF i-bound ts-i]
have consis: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb**.**
from dist [rule-format, OF i-bound] ts-i
obtain
dist-sb: $\forall$ a $\in$ all-acquired sb. $\mathcal{S}'$ a = $\mathcal{S}$ a
by auto

from sharing-consistent-shared-exchange [OF dist-sb consis]
show ?thesis **.**
qed
qed

lemma all-acquired-takeWhile: all-acquired (takeWhile P sb) $\subseteq$ all-acquired sb
proof –
from all-acquired-append [of takeWhile P sb dropWhile P sb]
show ?thesis
by auto
qed

lemma all-acquired-dropWhile: all-acquired (dropWhile P sb) $\subseteq$ all-acquired sb
proof –
from all-acquired-append [of takeWhile P sb dropWhile P sb]
show ?thesis
by auto
qed

lemma acquired-share-owns-shared:
assumes consis: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb
shows acquired pending-write sb $\mathcal{O}$ $\cup$ dom (share sb $\mathcal{S}$) $\subseteq$ $\mathcal{O}$ $\cup$ dom $\mathcal{S}$
proof –
from acquired-all-acquired **have** acquired pending-write sb $\mathcal{O}$ $\subseteq$ $\mathcal{O}$ $\cup$ all-acquired sb**.**
moreover
from sharing-consistent-all-acquired [OF consis] **have** all-acquired sb $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$**.**
moreover
from sharing-consistent-share-all-shared **have** dom (share sb $\mathcal{S}$) $\subseteq$ dom $\mathcal{S}$ $\cup$ all-shared sb**.**
moreover
from sharing-consistent-all-shared [OF consis] **have** all-shared sb $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$**.**
ultimately
show ?thesis
by blast
qed

lemma acquired-owns-shared:
assumes consis: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb
shows acquired True sb $\mathcal{O}$ $\subseteq$ $\mathcal{O}$ $\cup$ dom $\mathcal{S}$
using acquired-share-owns-shared [OF consis]
by blast

lemma share-owns-shared:
assumes consis: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb
shows dom (share sb $\mathcal{S}$) $\subseteq$ $\mathcal{O}$ $\cup$ dom $\mathcal{S}$
using acquired-share-owns-shared [OF consis]
by blast

lemma all-shared-append: all-shared (xs@ys) = all-shared xs ∪ all-shared ys
  by (induct xs) (auto split: memref.splits)

lemma acquired-union-notin-first:
  ⋀ pending-write A B. a ∈ acquired pending-write sb (A ∪ B) ⟹ a ∉ A ⟹ a ∈ acquired pending-write sb B
proof (induct sb)
  case Nil **thus** ?case **by** (auto split: if-split-asm)
next
  case (Cons x sb)
  then obtain a-notin-A: a ∉ A **and**
    a-acq: a ∈ acquired pending-write (x # sb) (A ∪ B)
    by blast
  show ?case
  proof (cases x)
    case ($\mathrm{Write}_{\mathsf{sb}}$ volatile a′ sop v A′ L R W)
    show ?thesis
    proof (cases volatile)
      case False
      with $\mathrm{Write}_{\mathsf{sb}}$ Cons **show** ?thesis **by** simp
    next
      case True
      note volatile = this
      show ?thesis
      proof (cases pending-write)
 case True
 from a-acq **have** a-acq′: a ∈ acquired True sb (A ∪ B ∪ A′ − R)
   by (simp add: $\mathrm{Write}_{\mathsf{sb}}$ volatile True)
 have (A ∪ B ∪ A′ − R) ⊆ (A ∪ (B ∪ A′ − R))
   by auto
 from acquired-mono-in [OF a-acq′ this]
 have a ∈ acquired True sb (A ∪ (B ∪ A′ − R))**.**
 from Cons.hyps [OF this a-notin-A]

 have a ∈ acquired True sb (B ∪ A′ − R)**.**
 then
 show ?thesis **by** (simp add: $\mathrm{Write}_{\mathsf{sb}}$ volatile True)
      next
 case False
 from a-acq **have** a-acq′: a ∈ acquired True sb (A′ − R)
   by (simp add: $\mathrm{Write}_{\mathsf{sb}}$ volatile False)
 then
 show ?thesis
   by (simp add: $\mathrm{Write}_{\mathsf{sb}}$ volatile False)
      qed
    qed
  next
    case ($\mathrm{Ghost}_{\mathsf{sb}}$ A′ L R W)
    show ?thesis
    proof (cases pending-write)

case True
from a-acq have a-acq′: a ∈ acquired True sb (A ∪ B ∪ A′ − R)
by (simp add: $\text{Ghost}_{\mathsf{sb}}$ True)
have (A ∪ B ∪ A′ − R) ⊆ (A ∪ (B ∪ A′ − R))
by auto
from acquired-mono-in [OF a-acq′ this]
have a ∈ acquired True sb (A ∪ (B ∪ A′ − R)).
from Cons.hyps [OF this a-notin-A]

have a ∈ acquired True sb (B ∪ A′ − R).
then
show ?thesis by (simp add: $\text{Ghost}_{\mathsf{sb}}$ True)
next
case False
from a-acq have a-acq′: a ∈ acquired False sb (A ∪ B)
by (simp add: $\text{Ghost}_{\mathsf{sb}}$ False)
from Cons.hyps [OF this a-notin-A]
show ?thesis
by (simp add: $\text{Ghost}_{\mathsf{sb}}$ False)
qed
qed (insert Cons, auto)
qed

lemma split-all-acquired-in:
a ∈ all-acquired xs ⟹
(∃sop a′ v ys zs A L R W. xs = ys @ $\text{Write}_{\mathsf{sb}}$ True a′ sop v A L R W# zs ∧ a ∈ A) ∨
(∃A L R W ys zs. xs = ys @ $\text{Ghost}_{\mathsf{sb}}$ A L R W# zs ∧ a ∈ A)
proof (induct xs)
case Nil thus ?case by simp
next
case (Cons x xs)
have a-in: a ∈ all-acquired (x # xs) by fact
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a′ sop v A L R W)
show ?thesis
proof (cases volatile)
case False
from a-in have a ∈ all-acquired xs
by (auto simp add: False $\text{Write}_{\mathsf{sb}}$)
from Cons.hyps [OF this]
have (∃sop a′ v ys zs A L R W. xs = ys @ $\text{Write}_{\mathsf{sb}}$ True a′ sop v A L R W# zs ∧ a
∈ A) ∨

($\exists$A L R W ys zs. xs = ys @ Ghost$_{sb}$ A L R W # zs $\wedge$ a $\in$ A) (**is** ?write $\vee$?ghst).
then
show ?thesis
proof
assume ?write
then
obtain sop$''$ a$''$ v$''$ A$''$ L$''$ R$''$ W$''$ ys zs
where xs=ys@Write$_{sb}$ True a$''$ sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$#zs **and** a-in: a $\in$ A$''$
by auto
hence x#xs = (x#ys)@Write$_{sb}$ True a$''$ sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$#zs
by auto
thus ?thesis
using a-in
by blast
next
assume ?ghst
then obtain A$''$ L$''$ R$''$ W$''$ ys zs **where**
xs=ys@Ghost$_{sb}$ A$''$ L$''$ R$''$ W$''$#zs **and** a-in: a $\in$ A$''$
by auto
hence x#xs = (x#ys)@Ghost$_{sb}$ A$''$ L$''$ R$''$ W$''$#zs
by auto
thus ?thesis
using a-in
by blast
qed
next
case True
note volatile = this
show ?thesis
proof (cases a $\in$ A)
case False
with a-in **have** a $\in$ all-acquired xs
by (auto simp add: volatile Write$_{sb}$)
from Cons.hyps [OF this]
have ($\exists$sop a$'$ v ys zs A L R W. xs = ys @ Write$_{sb}$ True a$'$ sop v A L R W # zs $\wedge$ a $\in$ A) $\vee$
($\exists$A L R W ys zs. xs = ys @ Ghost$_{sb}$ A L R W# zs $\wedge$ a $\in$ A) (**is** ?write $\vee$?ghst).
then
show ?thesis
proof
assume ?write
then
obtain sop$''$ a$''$ v$''$ A$''$ L$''$ R$''$ W$''$ ys zs
where xs=ys@Write$_{sb}$ True a$''$ sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$ #zs **and** a-in: a $\in$ A$''$
by auto
hence x#xs = (x#ys)@Write$_{sb}$ True a$''$ sop$''$ v$''$ A$''$ L$''$ R$''$ W$''$#zs
by auto
thus ?thesis
using a-in

by blast
next
assume ?ghst
then obtain A′′ L′′ R′′ W′′ ys zs **where**
xs=ys @Ghost$_{sb}$ A′′ L′′ R′′ W′′#zs **and** a-in: a ∈ A′′
by auto
hence x#xs = (x#ys)@Ghost$_{sb}$ A′′ L′′ R′′ W′′#zs
by auto
thus ?thesis
using a-in
by blast
qed
next
case True
then have x#xs=[]@(Write$_{sb}$ True a′ sop v A L R W#xs)
by (simp add: Write$_{sb}$ volatile True)
thus ?thesis
using True
by blast
qed
qed
next
case Read$_{sb}$
from a-in **have** a ∈ all-acquired xs
by (auto simp add: Read$_{sb}$)
from Cons.hyps [OF this]
have (∃sop a′ v ys zs A L R W. xs = ys @ Write$_{sb}$ True a′ sop v A L R W# zs ∧ a ∈ A) ∨
(∃A L R W ys zs. xs = ys @ Ghost$_{sb}$ A L R W# zs ∧ a ∈ A) (**is** ?write ∨ ?ghst)**.**
then
show ?thesis
proof
assume ?write
then
obtain sop′′ a′′ v′′ A′′ L′′ R′′ W′′ ys zs
where xs=ys@Write$_{sb}$ True a′′ sop′′ v′′ A′′ L′′ R′′ W′′#zs **and** a-in: a ∈ A′′
by auto
hence x#xs = (x#ys)@Write$_{sb}$ True a′′ sop′′ v′′ A′′ L′′ R′′ W′′#zs
by auto
thus ?thesis
using a-in
by blast
next
assume ?ghst
then obtain A′′ L′′ R′′ W′′ ys zs **where**
xs=ys@Ghost$_{sb}$ A′′ L′′ R′′ W′′#zs **and** a-in: a ∈ A′′
by auto
hence x#xs = (x#ys)@Ghost$_{sb}$ A′′ L′′ R′′ W′′#zs
by auto
thus ?thesis

```
using a-in
by blast
   qed
 next
   case Prog_sb
   from a-in have a ∈ all-acquired xs
     by (auto simp add: Prog_sb)
   from Cons.hyps [OF this]
   have (∃sop a′ v ys zs A L R W. xs = ys @ Write_sb True a′ sop v A L R W# zs ∧ a ∈
A) ∨
          (∃A L R W ys zs. xs = ys @ Ghost_sb A L R W# zs ∧ a ∈ A) (is ?write ∨ ?ghst).
   then
   show ?thesis
   proof
     assume ?write
     then
     obtain sop″ a″ v″ A″ L″ R″ W″ ys zs
 where xs=ys@Write_sb True a″ sop″ v″ A″ L″ R″ W″#zs and a-in: a ∈ A″
 by auto
     hence x#xs = (x#ys)@Write_sb True a″ sop″ v″ A″ L″ R″ W″#zs
 by auto
     thus ?thesis
 using a-in
 by blast
   next
     assume ?ghst
     then obtain A″ L″ R″ W″ ys zs where
 xs=ys@Ghost_sb A″ L″ R″ W″#zs and a-in: a ∈ A″
 by auto
     hence x#xs = (x#ys)@Ghost_sb A″ L″ R″ W″#zs
 by auto
     thus ?thesis
 using a-in
 by blast
   qed
 next
   case (Ghost_sb A L R W)
   show ?thesis
   proof (cases a ∈ A)
     case False
     with a-in have a ∈ all-acquired xs
 by (auto simp add: Ghost_sb)
     from Cons.hyps [OF this]
     have (∃sop a′ v ys zs A L R W. xs = ys @ Write_sb True a′ sop v A L R W # zs ∧ a
∈ A) ∨
          (∃A L R W ys zs. xs = ys @ Ghost_sb A L R W# zs ∧ a ∈ A) (is ?write ∨ ?ghst).
     then
     show ?thesis
     proof
 assume ?write
```

then
obtain sop'' a'' v'' A'' L'' R'' W'' ys zs
where xs=ys@Write$_{sb}$ True a'' sop'' v'' A'' L'' R'' W''#zs **and** a-in: a ∈ A''
by auto
hence x#xs = (x#ys)@Write$_{sb}$ True a'' sop'' v'' A'' L'' R'' W''#zs
by auto
thus ?thesis
using a-in
by blast
next
assume ?ghst
then obtain A'' L'' R'' W'' ys zs **where**
xs=ys@Ghost$_{sb}$ A'' L'' R'' W''#zs **and** a-in: a ∈ A''
by auto
hence x#xs = (x#ys)@Ghost$_{sb}$ A'' L'' R'' W''#zs
by auto
thus ?thesis
using a-in
by blast
qed
next
case True

then have x#xs=[]@(Ghost$_{sb}$ A L R W#xs)
by (simp add: Ghost$_{sb}$ True)
thus ?thesis
using True
by blast
qed
qed
qed

lemma split-Write$_{sb}$-in-outstanding-refs:
a ∈ outstanding-refs is-Write$_{sb}$ xs ⟹ (∃sop volatile v ys zs A L R W. xs = ys@(Write$_{sb}$ volatile a sop v A L R W#zs))
proof (induct xs)
case Nil **thus** ?case **by** simp
next
case (Cons x xs)
have a-in: a ∈ outstanding-refs is-Write$_{sb}$ (x # xs) **by** fact
show ?case
proof (cases x)
case (Write$_{sb}$ volatile a' sop v A L R W)
show ?thesis
proof (cases a'=a)
case False
with a-in **have** a ∈ outstanding-refs is-Write$_{sb}$ xs
by (auto simp add: Write$_{sb}$)
from Cons.hyps [OF this] **obtain** sop'' volatile'' v'' A'' L'' R'' W'' ys zs

```
 where xs=ys@Write_sb volatile′′ a sop′′ v′′ A′′ L′′ R′′ W′′#zs
 by auto
     hence x#xs = (x#ys)@Write_sb volatile′′ a sop′′ v′′ A′′ L′′ R′′ W′′#zs
 by auto
     thus ?thesis
 by blast
   next
     case True
     then have x#xs=[]@(Write_sb volatile a sop v A L R W#xs)
 by (simp add: Write_sb True)
     thus ?thesis
 by blast
   qed
  next
   case Read_sb
   from a-in have a ∈ outstanding-refs is-Write_sb xs
     by (auto simp add: Read_sb)
   from Cons.hyps [OF this] obtain sop′′ volatile′′ v′′ A′′ L′′ R′′ W′′ ys zs
     where xs=ys@Write_sb volatile′′ a sop′′ v′′ A′′ L′′ R′′ W′′ #zs
     by auto
   hence x#xs = (x#ys)@Write_sb volatile′′ a sop′′ v′′ A′′ L′′ R′′ W′′#zs
     by auto
   thus ?thesis
     by blast
  next
   case Prog_sb
   from a-in have a ∈ outstanding-refs is-Write_sb xs
     by (auto simp add: Prog_sb)
   from Cons.hyps [OF this] obtain sop′′ volatile′′ v′′ A′′ L′′ R′′ W′′ ys zs
     where xs=ys@Write_sb volatile′′ a sop′′ v′′ A′′ L′′ R′′ W′′#zs
     by auto
   hence x#xs = (x#ys)@Write_sb volatile′′ a sop′′ v′′ A′′ L′′ R′′ W′′#zs
     by auto
   thus ?thesis
     by blast
  next
   case Ghost_sb
   from a-in have a ∈ outstanding-refs is-Write_sb xs
     by (auto simp add: Ghost_sb)
   from Cons.hyps [OF this] obtain sop′′ volatile′′ v′′ A′′ L′′ R′′ W′′ ys zs
     where xs=ys@Write_sb volatile′′ a sop′′ v′′ A′′ L′′ R′′ W′′#zs
     by auto
   hence x#xs = (x#ys)@Write_sb volatile′′ a sop′′ v′′ A′′ L′′ R′′ W′′#zs
     by auto
   thus ?thesis
     by blast
  qed
qed
```

lemma outstanding-refs-is-Write$_{sb}$-union:

outstanding-refs is-Write$_{sb}$ xs =
(outstanding-refs is-volatile-Write$_{sb}$ xs ∪ outstanding-refs is-non-volatile-Write$_{sb}$ xs)
apply (induct xs)
apply simp
subgoal for a
apply (case-tac a)
apply auto
done
done

lemma rtranclp-r-rtranclp: ⟦r^{**} x y; r y z⟧ ⟹ r^{**} x z
by auto

lemma r-rtranclp-rtranclp: ⟦r x y; r^{**} y z⟧ ⟹ r^{**} x z
by auto

lemma unshared-is-non-volatile-Write$_{sb}$: $\bigwedge\mathcal{S}$.
⟦non-volatile-writes-unshared $\mathcal{S}$ sb; a ∈ dom $\mathcal{S}$; a ∉ all-unshared sb⟧ ⟹
a ∉ outstanding-refs is-non-volatile-Write$_{sb}$ sb
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case (Write$_{sb}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)
case False
with Cons Write$_{sb}$ **show** ?thesis **by** auto
next
case True
from Cons.hyps [**where** $\mathcal{S}$=($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)] Cons.prems
show ?thesis
by (auto simp add: Write$_{sb}$ True)
qed
next
case Read$_{sb}$ **with** Cons **show** ?thesis **by** auto
next
case Prog$_{sb}$ **with** Cons **show** ?thesis **by** auto
next
case (Ghost$_{sb}$ A L R W)
with Cons.hyps [**where** $\mathcal{S}$=($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)] Cons.prems **show** ?thesis **by** auto
qed
qed

lemma outstanding-non-volatile-Read$_{sb}$-acquired-or-read-only-reads:
$\bigwedge\mathcal{O}$ $\mathcal{S}$ pending-write.
⟦non-volatile-owned-or-read-only pending-write $\mathcal{S}$ $\mathcal{O}$ sb;

a ∈ outstanding-refs is-non-volatile-Read$_{sb}$ sb⟧
⟹ a ∈ acquired-reads True sb $\mathcal{O}$ ∨ a ∈ read-only-reads $\mathcal{O}$ sb
proof (induct sb)
  case Nil **thus** ?case **by** simp
next
  case (Cons x sb)
  show ?case
  proof (cases x)
    case (Write$_{sb}$ volatile a′ sop v A L R W)
    show ?thesis
    proof (cases volatile)
      case True
      with Write$_{sb}$ Cons.hyps [of True ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) ($\mathcal{O}$ ∪ A − R)] Cons.prems
      show ?thesis **by** auto
    next
      case False
      with Cons **show** ?thesis
 by (auto simp add: Write$_{sb}$)
    qed
  next
    case (Read$_{sb}$ volatile a′ t v)
    show ?thesis
    proof (cases volatile)
      case False **with** Read$_{sb}$ Cons **show** ?thesis **by** auto
    next
      case True
      with Read$_{sb}$ Cons **show** ?thesis **by** auto
    qed
  next
    case Prog$_{sb}$ **with** Cons **show** ?thesis **by** auto
  next
    case (Ghost$_{sb}$ A L R W) **with** Cons.hyps [of pending-write ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) $\mathcal{O}$ ∪ A − R] Cons.prems
    show ?thesis
      by auto
  qed
qed

lemma acquired-reads-union: ⋀pending-writes A B.
  ⟦a ∈ acquired-reads pending-writes sb (A ∪ B); a ∉ A⟧ ⟹ a ∈ acquired-reads pending-writes sb B
proof (induct sb)
  case Nil **thus** ?case **by** simp
next
  case (Cons x sb)
  show ?case
  proof (cases x)
    case (Write$_{sb}$ volatile a′ sop v A′ L′ R′ W′)
    show ?thesis
    proof (cases volatile)

```
    case True
    note volatile=this
    show ?thesis
    proof (cases pending-writes)
case True
from Cons.prems obtain
 a-in: a ∈ acquired-reads True sb (A ∪ B ∪ A′ − R′) and
 a-notin: a ∉ A
 by (simp add: Write_sb volatile True)
have (A ∪ B ∪ A′ − R′) ⊆  (A ∪ (B ∪ A′ − R′))
 by auto
from acquired-reads-mono [OF this ] a-in
have a ∈ acquired-reads True sb (A ∪ (B ∪ A′ − R′))
 by auto

from Cons.hyps [OF this a-notin]
have a ∈ acquired-reads True sb (B ∪ A′ − R′).
then show ?thesis
 by (simp add: Write_sb volatile True)
    next
case False
with Cons show ?thesis
 by (auto simp add: Write_sb volatile False)
    qed
   next
    case False
    with Cons show ?thesis
by (auto simp add: Write_sb False)
  qed
 next
  case Read_sb with Cons show ?thesis
    by (auto split: if-split-asm)
 next
  case Prog_sb with Cons show ?thesis
    by (auto)
 next
  case (Ghost_sb A′ L′ R′ W′)
  show ?thesis
  proof −
    from Cons.prems obtain
a-in: a ∈ acquired-reads pending-writes sb (A ∪ B ∪ A′ − R′) and
a-notin: a ∉ A
     by (simp add: Ghost_sb )
    have (A ∪ B ∪ A′ − R′) ⊆  (A ∪ (B ∪ A′ − R′))
     by auto
    from acquired-reads-mono [OF this ] a-in
    have a ∈ acquired-reads pending-writes sb (A ∪ (B ∪ A′ − R′))
     by auto

    from Cons.hyps [OF this a-notin]
```

have a $\in$ acquired-reads pending-writes sb (B $\cup$ A$'$ $-$ R$'$)**.**
then show ?thesis
by (simp add: Ghost$_{\mathsf{sb}}$)
qed
qed
qed

lemma non-volatile-writes-unshared-no-outstanding-non-volatile-Write$_{\mathsf{sb}}$: $\bigwedge\mathcal{S}\ \mathcal{S}'$.
$[\![$non-volatile-writes-unshared $\mathcal{S}$ sb;
$\forall$ a $\in$ dom $\mathcal{S}'$ $-$ dom $\mathcal{S}$. a $\notin$ outstanding-refs is-non-volatile-Write$_{\mathsf{sb}}$ sb $]\!]$
$\Longrightarrow$ non-volatile-writes-unshared $\mathcal{S}'$ sb
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case (Write$_{\mathsf{sb}}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)
case True
from Cons.prems **obtain**
unshared-sb: non-volatile-writes-unshared ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) sb **and**
no-refs-sb: $\forall$ a$\in$dom $\mathcal{S}'$ $-$ dom $\mathcal{S}$. a $\notin$ outstanding-refs is-non-volatile-Write$_{\mathsf{sb}}$ sb
by (simp add: Write$_{\mathsf{sb}}$ True)
from no-refs-sb **have** $\forall$ a$\in$dom ($\mathcal{S}'$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $-$ dom ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L).
a $\notin$ outstanding-refs is-non-volatile-Write$_{\mathsf{sb}}$ sb
by auto
from Cons.hyps [OF unshared-sb this]
show ?thesis
by (simp add: Write$_{\mathsf{sb}}$ True)
next
case False
with Cons **show** ?thesis
by (auto simp add: Write$_{\mathsf{sb}}$ False)
qed
next
case Read$_{\mathsf{sb}}$ **with** Cons **show** ?thesis
by (auto)
next
case Prog$_{\mathsf{sb}}$ **with** Cons **show** ?thesis
by (auto)
next
case (Ghost$_{\mathsf{sb}}$ A L R W)
from Cons.prems **obtain**
unshared-sb: non-volatile-writes-unshared ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) sb **and**
no-refs-sb: $\forall$ a$\in$dom $\mathcal{S}'$ $-$ dom $\mathcal{S}$. a $\notin$ outstanding-refs is-non-volatile-Write$_{\mathsf{sb}}$ sb
by (simp add: Ghost$_{\mathsf{sb}}$)
from no-refs-sb **have** $\forall$ a$\in$dom ($\mathcal{S}'$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $-$ dom ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L).

$a \notin$ outstanding-refs is-non-volatile-Write$_{sb}$ sb
by auto
from Cons.hyps [OF unshared-sb this]
show ?thesis
by (simp add: Ghost$_{sb}$)
qed
qed

theorem sharing-consis-share-all-until-volatile-write:
$\bigwedge\mathcal{S}$ ts$'$. $[\![$ownership-distinct ts; sharing-consis $\mathcal{S}$ ts; length ts$'$ = length ts;
$\forall$ i < length ts.
(let (-,-,-,sb,-,$\mathcal{O}$,-) = ts!i;
(-,-,-,sb$'$,-,$\mathcal{O}'$,-) = ts$'$!i
in $\mathcal{O}'$ = acquired True (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\mathcal{O}$ $\wedge$
sb$'$ = dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb)$]\!]$ $\Longrightarrow$
sharing-consis (share-all-until-volatile-write ts $\mathcal{S}$) ts$'$ $\wedge$
dom (share-all-until-volatile-write ts $\mathcal{S}$) $-$ dom $\mathcal{S}$ $\subseteq$
$\bigcup$ ((λ(-,-,-,-,-,$\mathcal{O}$,-). $\mathcal{O}$) ‘ set ts) $\wedge$
dom $\mathcal{S}$ $-$ dom (share-all-until-volatile-write ts $\mathcal{S}$) $\subseteq$
$\bigcup$ ((λ(-,-,-,sb,-,$\mathcal{O}$,-). all-acquired sb $\cup$ $\mathcal{O}$) ‘ set ts)
proof (induct ts)
case Nil **thus** ?case **by** auto
next
case (Cons t ts)
have leq: length ts$'$ = length (t#ts) **by** fact
have sim: $\forall$ i < length (t#ts).
(let (-,-,-,sb,-,$\mathcal{O}$,-) = (t#ts)!i;
(-,-,-,sb$'$,-,$\mathcal{O}'$,-) = ts$'$!i
in $\mathcal{O}'$ = acquired True (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\mathcal{O}$ $\wedge$
sb$'$ = dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb)
by fact
obtain p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
where t: t = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by (cases t)

from leq **obtain** t$'$ ts$''$ **where** ts$'$: ts$'$=t$'$#ts$''$ **and** leq$'$: length ts$''$ = length ts
by (cases ts$'$) force+

obtain p$'$ is$'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{D}'$ j$'$ sb$'$
where t$'$: t$'$ = (p$'$,is$'$,j$'$,sb$'$,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)
by (cases t$'$)

from sim [rule-format, of 0] t t$'$ ts$'$
obtain $\mathcal{O}'$: $\mathcal{O}'$ = acquired True (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\mathcal{O}$ **and**
sb$'$: sb$'$ = dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb
by auto

from sim ts$'$

have sim$'$: $\forall$ i < length ts.
(let (-,-,-,sb,-,$\mathcal{O}$,$\mathcal{R}$) = ts!i;
(-,-,-,sb$'$,-,$\mathcal{O}'$,$\mathcal{R}$) = ts$''$!i
in $\mathcal{O}'$ = acquired True (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\mathcal{O}$ $\wedge$
sb$'$ = dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb)
by auto

have consis: sharing-consis $\mathcal{S}$ (t#ts) **by** fact
then interpret sharing-consis $\mathcal{S}$ (t#ts)**.**
from sharing-consis [of 0] t
have consis-sb: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb
by fastforce
from sharing-consistent-takeWhile [OF this]
have consis$'$: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb)
by simp

let ?$\mathcal{S}'$ = (share (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\mathcal{S}$)
from freshly-shared-owned [OF consis$'$]
have fresh-owned: dom ?$\mathcal{S}'$ $-$ dom $\mathcal{S}$ $\subseteq$ $\mathcal{O}$**.**
from unshared-all-unshared [OF consis$'$] unshared-acquired-or-owned [OF consis$'$]
have unshared-acq-owned: dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$
$\subseteq$ all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\cup$ $\mathcal{O}$
by simp

have dist: ownership-distinct (t#ts) **by** fact
from ownership-distinct-tl [OF this]
have dist$'$: ownership-distinct ts **.**

from sharing-consis-tl [OF consis]
interpret consis$'$: sharing-consis $\mathcal{S}$ ts**.**

from dist **interpret** ownership-distinct (t#ts)**.**

have sep:
$\forall$ i < length ts. let (-,-,-,sb$'$,-,-,-) = ts!i in
all-acquired sb$'$ $\cap$ dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$ = {} $\wedge$
all-unshared sb$'$ $\cap$ dom ?$\mathcal{S}'$ $-$ dom $\mathcal{S}$ = {}
proof $-$
{
fix i p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j_i sb_i
assume i-bound: i < length ts
assume ts-i: ts ! i = (p_i,is_i,j_i,sb_i,$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$)
have all-acquired sb_i $\cap$ dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$ = {} $\wedge$
all-unshared sb_i $\cap$ dom ?$\mathcal{S}'$ $-$ dom $\mathcal{S}$ = {}
proof $-$

from ownership-distinct [of 0 Suc i] ts-i t i-bound
have dist: $(\mathcal{O} \cup \text{all-acquired sb}) \cap (\mathcal{O}_i \cup \text{all-acquired sb}_i) = \{\}$
  by force

from dist unshared-acq-owned all-acquired-takeWhile [of (Not $\circ$ is-volatile-Write$_{sb}$) sb]
have all-acquired $\text{sb}_i \cap \text{dom } \mathcal{S} - \text{dom } ?\mathcal{S}' = \{\}$
  by blast

moreover

from sharing-consis [of Suc i] ts-i i-bound
have sharing-consistent $\mathcal{S}$ $\mathcal{O}_i$ sb_i
  by force
from unshared-acquired-or-owned [OF this]
have all-unshared $\text{sb}_i \subseteq \text{all-acquired sb}_i \cup \mathcal{O}_i$.
with dist fresh-owned
have all-unshared $\text{sb}_i \cap \text{dom } ?\mathcal{S}' - \text{dom } \mathcal{S} = \{\}$
  by blast

ultimately show ?thesis **by** simp
    qed
  }
  thus ?thesis
    by (fastforce simp add: Let-def)
qed

from consis$'$.sharing-consis-preservation [OF sep]
have consis-ts: sharing-consis $?\mathcal{S}'$ ts.

from Cons.hyps [OF dist$'$ this leq$'$ sim$'$]
obtain consis-ts$''$:
  sharing-consis (share-all-until-volatile-write ts $?\mathcal{S}'$) ts$''$ **and**

  fresh: dom (share-all-until-volatile-write ts $?\mathcal{S}'$) $-$ dom $?\mathcal{S}' \subseteq$
       $\bigcup$ ((λ(-,-,-,-,-,$\mathcal{O}$,$\mathcal{R}$). $\mathcal{O}$) ‘ set ts) **and**
  unshared: dom $?\mathcal{S}'$ $-$ dom (share-all-until-volatile-write ts $?\mathcal{S}'$) $\subseteq$
       $\bigcup$ ((λ(-,-,-,sb,-,$\mathcal{O}$,$\mathcal{R}$). all-acquired sb $\cup$ $\mathcal{O}$)‘ set ts)
  by auto

from sharing-consistent-append [of - - (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb)
(dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb)] consis-sb
have consis-t$'$: sharing-consistent $?\mathcal{S}'$ $\mathcal{O}'$ sb$'$
  by (simp add: $\mathcal{O}'$ sb$'$)

have fresh-dist: all-acquired sb$'$ ∩ dom ?$\mathcal{S}'$ − dom (share-all-until-volatile-write ts ?$\mathcal{S}'$) = {}
proof −
have all-acquired sb$'$ ∩ $\bigcup$ ((λ(-,-,-,sb,-,$\mathcal{O}$,-). all-acquired sb ∪ $\mathcal{O}$)' set ts) = {}
proof −
{
fix x
assume x-sb$'$: x ∈ all-acquired sb$'$
assume x-ts: x ∈ $\bigcup$ ((λ(-,-,-,sb,-,$\mathcal{O}$,-). all-acquired sb ∪ $\mathcal{O}$)' set ts)
have False
proof −
from x-ts
obtain i p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j_i sb_i **where**
i-bound: i < length ts **and**
ts-i: ts!i = (p_i,is_i,j_i,sb_i,$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$) **and**
x-in: x ∈ all-acquired sb_i ∪ $\mathcal{O}_i$
by (force simp add: in-set-conv-nth)
from ownership-distinct [of 0 Suc i] ts-i t i-bound
have dist: ($\mathcal{O}$ ∪ all-acquired sb) ∩ ($\mathcal{O}_i$ ∪ all-acquired sb_i) = {}
by force
with x-sb$'$ x-in all-acquired-dropWhile [of (Not ∘ is-volatile-Write$_{sb}$) sb] **show** False
by (auto simp add: sb$'$)
qed
}
thus ?thesis **by** blast
qed
with unshared **show** ?thesis
by blast
qed

have unshared-dist: all-unshared sb$'$ ∩ dom (share-all-until-volatile-write ts ?$\mathcal{S}'$) − dom ?$\mathcal{S}'$ = {}
proof −
from unshared-acquired-or-owned [OF consis-t$'$]
have all-unshared sb$'$ ⊆ all-acquired sb$'$ ∪ $\mathcal{O}'$**.**
also
from all-acquired-dropWhile [of (Not ∘ is-volatile-Write$_{sb}$) sb]
acquired-all-acquired [of True takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb $\mathcal{O}$]
all-acquired-takeWhile [of (Not ∘ is-volatile-Write$_{sb}$) sb]
have all-acquired sb$'$ ∪ $\mathcal{O}'$ ⊆ all-acquired sb ∪ $\mathcal{O}$
by (auto simp add: sb$'$ $\mathcal{O}'$)
finally
have all-unshared sb$'$ ⊆ (all-acquired sb ∪ $\mathcal{O}$)**.**

moreover

have (all-acquired sb ∪ $\mathcal{O}$) ∩ $\bigcup$ ((λ(-,-,-,-,-,$\mathcal{O}$,-). $\mathcal{O}$) ' set ts) = {}
proof −
{
fix x

assume x-sb′: x ∈ all-acquired sb ∪ $\mathcal{O}$
assume x-ts: x ∈ $\bigcup$ ((λ(-,-,-,-,-,$\mathcal{O}$,-). $\mathcal{O}$)‘ set ts)
have False
proof −
from x-ts
obtain i p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j_i sb_i **where**
i-bound: i < length ts **and**
ts-i: ts!i = (p_i,is_i,j_i,sb_i,$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$) **and**
x-in: x ∈ $\mathcal{O}_i$
by (force simp add: in-set-conv-nth)
from ownership-distinct [of 0 Suc i] ts-i t i-bound
have dist: ($\mathcal{O}$ ∪ all-acquired sb) ∩ ($\mathcal{O}_i$ ∪ all-acquired sb_i) = {}
by force
with x-sb′ x-in **show** False
by (auto simp add: sb′)
qed
}
thus ?thesis **by** blast
qed
ultimately show ?thesis
using fresh **by** fastforce
qed

from sharing-consistent-preservation [OF consis-t′ fresh-dist unshared-dist]
have consis-ts: sharing-consistent (share-all-until-volatile-write ts ?$\mathcal{S}$′) $\mathcal{O}$′ sb′**.**
note sharing-consis-Cons [OF consis-ts″ consis-ts, of p′ is′ j′ $\mathcal{D}$′]
moreover
from fresh fresh-owned
have dom (share-all-until-volatile-write ts ?$\mathcal{S}$′) − dom $\mathcal{S}$ ⊆
$\mathcal{O}$ ∪ $\bigcup$ ((λ(-,-,-,-,-,$\mathcal{O}$,-). $\mathcal{O}$) ‘ set ts)
by auto
moreover
from unshared unshared-acq-owned all-acquired-takeWhile [of (Not ∘ is-volatile-Write$_{sb}$) sb]
have dom $\mathcal{S}$ − dom (share-all-until-volatile-write ts ?$\mathcal{S}$′) ⊆
all-acquired sb ∪ $\mathcal{O}$ ∪ $\bigcup$ ((λ(-,-,-,sb,-,$\mathcal{O}$,-). all-acquired sb ∪ $\mathcal{O}$) ‘ set ts)
by auto
ultimately

show ?case
by (auto simp add: t ts′ t′)
qed

corollary sharing-consistent-share-all-until-volatile-write:
assumes dist: ownership-distinct ts
assumes consis: sharing-consis $\mathcal{S}$ ts
assumes i-bound: i < length ts
assumes ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
shows sharing-consistent (share-all-until-volatile-write ts $\mathcal{S}$)

(acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{O}$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
proof −
define ts′ **where** ts′ == map (λ(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$).
(p,is,j,
dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb,$\mathcal{D}$,acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{O}$,$\mathcal{R}$)) ts
have leq: length ts′ = length ts
by (simp add: ts′-def)

have flush: ∀ i < length ts.
(let (-,-,-,sb,-,$\mathcal{O}$,-) = ts!i;
(-,-,-,sb′,-,$\mathcal{O}$′,-) = ts′!i
in $\mathcal{O}$′ = acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{O}$ ∧
sb′ = dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
by (auto simp add: ts′-def Let-def)

from sharing-consis-share-all-until-volatile-write [OF dist consis leq flush]
interpret sharing-consis (share-all-until-volatile-write ts $\mathcal{S}$) ts′ **by** simp
from i-bound leq ts-i sharing-consis [of i]
show ?thesis
by (force simp add: ts′-def)
qed

lemma restrict-map-UNIV [simp]: S |‘ UNIV = S
by (auto simp add: restrict-map-def)

lemma share-all-until-volatile-write-Read-commute:
shows ⋀S i. ⟦i < length ls; ls!i=(p,Read volatile a t#is,j,sb,$\mathcal{D}$,$\mathcal{O}$)
⟧
⟹
share-all-until-volatile-write
(ls[i := (p,is, j(t↦v), sb @ [Read$_{sb}$ volatile a t v],$\mathcal{D}$′, $\mathcal{O}$)]) S =
share-all-until-volatile-write ls S
proof (induct ls)
case Nil **thus** ?case
by simp
next
case (Cons l ls)
note i-bound = ‹i < length (l#ls)›
note ith = ‹(l#ls)!i = (p,Read volatile a t#is,j,sb,$\mathcal{D}$,$\mathcal{O}$)›
show ?case
proof (cases i)
case 0

from ith 0 **have** l: l = (p,Read volatile a t#is,j,sb,$\mathcal{D}$,$\mathcal{O}$)
  by simp
thus ?thesis
  by (simp add: 0 share-append-Read$_{\mathsf{sb}}$ del: fun-upd-apply)
next
  case (Suc n)
  obtain p_l is_l $\mathcal{O}_l$ $\mathcal{D}_l$ j_l sb_l **where** l: l = (p_l,is_l,j_l,sb_l,$\mathcal{D}_l$,$\mathcal{O}_l$)
    by (cases l)
  from i-bound ith
  have share-all-until-volatile-write
    (ls[n := (p,is, j(t$\mapsto$v), sb @ [Read$_{\mathsf{sb}}$ volatile a t v],$\mathcal{D}'$, $\mathcal{O}$)])
    (share (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb_l) S) =
    share-all-until-volatile-write ls (share (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb_l) S)
    apply −
    apply (rule Cons.hyps)
    apply (auto simp add: Suc l)
    done

  then
  show ?thesis
    by (simp add: Suc l del: fun-upd-apply)
  qed
qed

lemma share-all-until-volatile-write-Write-commute:
  shows $\bigwedge$S i. ⟦i < length ls; ls!i=(p,Write volatile a (D,f) A L R W#is,j,sb,$\mathcal{D}$,$\mathcal{O}$)
  ⟧
  $\Longrightarrow$
  share-all-until-volatile-write
    (ls[i := (p,is,j, sb @ [Write$_{\mathsf{sb}}$ volatile a t (f j) A L R W], $\mathcal{D}'$, $\mathcal{O}$)]) S =
  share-all-until-volatile-write ls S
proof (induct ls)
  case Nil **thus** ?case
    by simp
next
  case (Cons l ls)
  note i-bound = ‹i < length (l#ls)›
  note ith = ‹(l#ls)!i = (p,Write volatile a (D,f) A L R W#is,j,sb,$\mathcal{D}$,$\mathcal{O}$)›
  show ?case
  proof (cases i)
    case 0
    from ith 0 **have** l: l = (p,Write volatile a (D,f) A L R W#is,j,sb,$\mathcal{D}$,$\mathcal{O}$)
      by simp
    thus ?thesis
      by (simp add: 0 share-append-Write$_{\mathsf{sb}}$ del: fun-upd-apply)
  next
    case (Suc n)
    obtain p_l is_l $\mathcal{O}_l$ $\mathcal{D}_l$ j_l sb_l **where** l: l = (p_l,is_l,j_l,sb_l,$\mathcal{D}_l$,$\mathcal{O}_l$)
      by (cases l)
    from i-bound ith

have share-all-until-volatile-write
(ls[n := (p,is, j, sb @ [Write$_{sb}$ volatile a t (f j) A L R W],$\mathcal{D}'$, $\mathcal{O}$)])
(share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_l$) S) =
share-all-until-volatile-write ls (share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_l$) S)
apply −
apply (rule Cons.hyps)
apply (auto simp add: Suc l)
done

then
show ?thesis
by (simp add: Suc l del: fun-upd-apply)
qed
qed

lemma share-all-until-volatile-write-RMW-commute:
shows $\bigwedge$S i. ⟦i < length ls; ls!i=(p,RMW a t (D,f) cond ret A L R W#is,j,[],$\mathcal{D}$,$\mathcal{O}$)
⟧
$\Longrightarrow$
share-all-until-volatile-write (ls[i := (p′,is, j′, [],$\mathcal{D}'$, $\mathcal{O}'$)]) S =
share-all-until-volatile-write ls S
proof (induct ls)
case Nil **thus** ?case
by simp
next
case (Cons l ls)
note i-bound = ‹i < length (l#ls)›
note ith = ‹(l#ls)!i = (p,RMW a t (D,f) cond ret A L R W#is,j,[],$\mathcal{D}$,$\mathcal{O}$)›
show ?case
proof (cases i)
case 0
from ith 0 **have** l: l = (p,RMW a t (D,f) cond ret A L R W#is,j,[],$\mathcal{D}$,$\mathcal{O}$)
by simp
thus ?thesis
by (simp add: 0 share-append-Write$_{sb}$ del: fun-upd-apply)
next
case (Suc n)
obtain p$_l$ is$_l$ $\mathcal{O}_l$ $\mathcal{D}_l$ j$_l$ sb$_l$ **where** l: l = (p$_l$,is$_l$,j$_l$,sb$_l$,$\mathcal{D}_l$,$\mathcal{O}_l$)
by (cases l)
from i-bound ith
have share-all-until-volatile-write
(ls[n := (p′,is,j′, [], $\mathcal{D}'$, $\mathcal{O}'$)])
(share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_l$) S) =
share-all-until-volatile-write ls (share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_l$) S)
apply −
apply (rule Cons.hyps)
apply (auto simp add: Suc l)
done

then

show ?thesis
by (simp add: Suc l del: fun-upd-apply)
qed
qed

lemma share-all-until-volatile-write-Fence-commute:
shows $\bigwedge$S i. ⟦i < length ls; ls!i=(p,Fence#is,j,[],$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
⟧
$\Longrightarrow$
share-all-until-volatile-write (ls[i := (p,is,j, [], $\mathcal{D}'$, $\mathcal{O}$,$\mathcal{R}'$)]) S =
share-all-until-volatile-write ls S
proof (induct ls)
case Nil **thus** ?case
by simp
next
case (Cons l ls)
note i-bound = ‹i < length (l#ls)›
note ith = ‹(l#ls)!i = (p,Fence#is,j,[],$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)›
show ?case
proof (cases i)
case 0
from ith 0 **have** l: l = (p,Fence#is,j,[],$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by simp
thus ?thesis
by (simp add: 0 share-append-Write$_{sb}$ del: fun-upd-apply)
next
case (Suc n)
obtain p_l is_l $\mathcal{O}_l$ $\mathcal{R}_l$ $\mathcal{D}_l$ j_l sb_l **where** l: l = (p_l,is_l,j_l,sb_l,$\mathcal{D}_l$,$\mathcal{O}_l$,$\mathcal{R}_l$)
by (cases l)
from i-bound ith
have share-all-until-volatile-write
(ls[n := (p,is, j, [],$\mathcal{D}'$, $\mathcal{O}$,$\mathcal{R}'$)])
(share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_l) S) =
share-all-until-volatile-write ls (share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_l) S)
apply −
apply (rule Cons.hyps)
apply (auto simp add: Suc l)
done

then
show ?thesis
by (simp add: Suc l del: fun-upd-apply)
qed
qed

lemma unshared-share-in: $\bigwedge$S. a $\in$ dom S $\Longrightarrow$ a $\notin$ all-unshared sb $\Longrightarrow$ a $\in$ dom (share sb S)
proof (induct sb)
  case Nil **thus** ?case **by** simp
next
 case (Cons x sb)
 show ?case
 proof (cases x)
  case (Write$_{sb}$ volatile a$'$ sop v A L R W)
  show ?thesis
  proof (cases volatile)
   case True
   show ?thesis
   proof −
 from Cons.prems **obtain** a-S: a $\in$ dom S **and** a-L: a $\notin$ L **and** a-sb: a $\notin$ all-unshared sb
  by (clarsimp simp add: Write$_{sb}$ True)
 from a-S a-L **have** a $\in$ dom (S $\oplus_{W}$ R $\ominus_{A}$ L)
  by auto
 from Cons.hyps [OF this a-sb]
 show ?thesis
  by (clarsimp simp add: Write$_{sb}$ True)
   qed
  next
   case False
   with Cons **show** ?thesis
 by (auto simp add: Write$_{sb}$ False)
  qed
 next
  case Read$_{sb}$
  with Cons **show** ?thesis
   by (auto simp add: Read$_{sb}$)
 next
  case Prog$_{sb}$
  with Cons **show** ?thesis
   by (auto simp add: Read$_{sb}$)
 next
  case Ghost$_{sb}$
  with Cons **show** ?thesis
   by (auto simp add: Ghost$_{sb}$)
 qed
qed

lemma dom-eq-dom-share-eq: $\bigwedge$S S$'$. dom S = dom S$'$ $\Longrightarrow$ dom (share sb S) = dom (share sb S$'$)
proof (induct sb)

case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case ($Write_{sb}$ volatile a' sop v A' L R W)
show ?thesis
proof (cases volatile)
case True
from Cons.prems
have dom ($S \oplus_W R \ominus_{A'} L$) = dom ($S' \oplus_W R \ominus_{A'} L$)
by auto
from Cons.hyps [OF this]
show ?thesis
by (clarsimp simp add: $Write_{sb}$ True)
next
case False **with** Cons.hyps [of S S'] Cons.prems $Write_{sb}$ **show** ?thesis **by** auto
qed
next
case $Read_{sb}$ **with** Cons.hyps [of S S'] Cons.prems **show** ?thesis **by** auto
next
case $Prog_{sb}$ **with** Cons.hyps [of S S'] Cons.prems **show** ?thesis **by** auto
next
case ($Ghost_{sb}$ A' L R W)
from Cons.prems
have dom ($S \oplus_W R \ominus_{A'} L$) = dom ($S' \oplus_W R \ominus_{A'} L$)
by auto
from Cons.hyps [OF this]
show ?thesis
by (clarsimp simp add: $Ghost_{sb}$)
qed
qed

lemma share-union:
$\bigwedge$A B. $[\![$a $\in$ dom (share sb ($A \oplus_z B$)); a $\notin$ dom A$]\!]$ $\Longrightarrow$ a $\in$ dom (share sb (Map.empty $\oplus_z$ B))
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case ($Write_{sb}$ volatile a' sop v A' L R W)
show ?thesis
proof (cases volatile)
case True
from Cons.prems
obtain a-in: a $\in$ dom (share sb (($A \oplus_z B$) $\oplus_W R \ominus_{A'} L$)) **and** a-A: a $\notin$ dom A
by (clarsimp simp add: $Write_{sb}$ True)
have dom (($A \oplus_z B$) $\oplus_W R \ominus_{A'} L$) $\subseteq$ dom ($A \oplus_z (B \cup R - L)$)

by auto

  from share-mono [OF this] a-in

  have a $\in$ dom (share sb (A $\oplus_{\mathsf{z}}$ (B $\cup$ R $-$ L)))

by blast

  from Cons.hyps [OF this] a-A

  have a $\in$ dom (share sb (Map.empty $\oplus_{\mathsf{z}}$ (B $\cup$ R $-$ L)))

by blast

  moreover

  have dom (Map.empty $\oplus_{\mathsf{z}}$ B $\cup$ R $-$ L) = dom ((Map.empty $\oplus_{\mathsf{z}}$ B) $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}'}$ L)

by auto

  note dom-eq-dom-share-eq [OF this, of sb]

  ultimately

  show ?thesis

by (clarsimp simp add: Write$_{\mathsf{sb}}$ True)

 next

  case False

  with Cons **show** ?thesis

by (auto simp add: Write$_{\mathsf{sb}}$ False)

 qed

next

 case Read$_{\mathsf{sb}}$

 with Cons **show** ?thesis

  by (auto simp add: Read$_{\mathsf{sb}}$)

next

 case Prog$_{\mathsf{sb}}$

 with Cons **show** ?thesis

  by (auto simp add: Read$_{\mathsf{sb}}$)

next

 case (Ghost$_{\mathsf{sb}}$ A$'$ L R W)

 from Cons.prems

 obtain a-in: a $\in$ dom (share sb ((A $\oplus_{\mathsf{z}}$ B) $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}'}$ L)) **and** a-A: a $\notin$ dom A

  by (clarsimp simp add: Ghost$_{\mathsf{sb}}$)

 have dom ((A $\oplus_{\mathsf{z}}$ B) $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}'}$ L) $\subseteq$ dom (A $\oplus_{\mathsf{z}}$ (B $\cup$ R $-$ L))

  by auto

 from share-mono [OF this] a-in

 have a $\in$ dom (share sb (A $\oplus_{\mathsf{z}}$ (B $\cup$ R $-$ L)))

  by blast

 from Cons.hyps [OF this] a-A

 have a $\in$ dom (share sb (Map.empty $\oplus_{\mathsf{z}}$ (B $\cup$ R $-$ L)))

  by blast

 moreover

 have dom (Map.empty $\oplus_{\mathsf{z}}$ B $\cup$ R $-$ L) = dom ((Map.empty $\oplus_{\mathsf{z}}$ B) $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}'}$ L)

  by auto

 note dom-eq-dom-share-eq [OF this, of sb]

 ultimately

 show ?thesis

  by (clarsimp simp add: Ghost$_{\mathsf{sb}}$)

qed

qed

lemma share-unshared-in:
$\bigwedge$S. a $\in$ dom (share sb S) $\Longrightarrow$ a $\in$ dom (share sb Map.empty) $\vee$ (a $\in$ dom S $\wedge$ a $\notin$ all-unshared sb)
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case (Write$_{sb}$ volatile a$'$ sop v A L R W)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems
have a-in: a $\in$ dom (share sb (S $\oplus_{W}$ R $\ominus_{A}$ L))
by (clarsimp simp add: Write$_{sb}$ True)
show ?thesis
proof (cases a $\in$ dom S)
case True
from Cons.hyps [OF a-in]
have a $\in$ dom (share sb Map.empty) $\vee$ a $\in$ dom (S $\oplus_{W}$ R $\ominus_{A}$ L) $\wedge$ a $\notin$ all-unshared sb**.**
then show ?thesis
proof
assume a $\in$ dom (share sb Map.empty)
from share-mono-in [OF this]
have a $\in$ dom (share sb (Map.empty $\oplus_{W}$ R $\ominus_{A}$ L)) **by** auto
then show ?thesis
by (clarsimp simp add: Write$_{sb}$ volatile True)
next
assume a $\in$ dom (S $\oplus_{W}$ R $\ominus_{A}$ L) $\wedge$ a $\notin$ all-unshared sb
then obtain a $\notin$ L a $\notin$ all-unshared sb
by auto
then show ?thesis **by** (clarsimp simp add: Write$_{sb}$ volatile True)
qed
next
case False
have dom (S $\oplus_{W}$ R $\ominus_{A}$ L) $\subseteq$ dom (S $\oplus_{W}$ (R $-$ L))
by auto
from share-mono [OF this] a-in
have a $\in$ dom (share sb (S $\oplus_{W}$ (R $-$ L))) **by** blast
from share-union [OF this False]
have a $\in$ dom (share sb (Map.empty $\oplus_{W}$ (R $-$ L)))**.**
moreover
have dom (Map.empty $\oplus_{W}$ (R $-$ L)) = dom (Map.empty $\oplus_{W}$ R $\ominus_{A}$ L)
by auto
note dom-eq-dom-share-eq [OF this, of sb]
ultimately
show ?thesis

```
  by (clarsimp simp add: Write_sb True)
     qed
    next
     case False
     with Cons show ?thesis
 by (auto simp add: Write_sb False)
    qed
  next
    case Read_sb
    with Cons show ?thesis
      by (auto simp add: Read_sb)
  next
    case Prog_sb
    with Cons show ?thesis
      by (auto simp add: Read_sb)
  next
    case (Ghost_sb A L R W)
    from Cons.prems
    have a-in: a ∈ dom (share sb (S ⊕_W R ⊖_A L))
      by (clarsimp simp add: Ghost_sb)
    show ?thesis
    proof (cases a ∈ dom S)
      case True
      from Cons.hyps [OF a-in]
      have a ∈ dom (share sb Map.empty) ∨ a ∈ dom (S ⊕_W R ⊖_A L) ∧ a ∉ all-unshared
sb.
      then show ?thesis
      proof
 assume a ∈ dom (share sb Map.empty)
 from share-mono-in [OF this]
 have a ∈ dom (share sb (Map.empty ⊕_W R ⊖_A L)) by auto
       then show ?thesis
  by (clarsimp simp add: Ghost_sb True)
      next
 assume a ∈ dom (S ⊕_W R ⊖_A L) ∧ a ∉ all-unshared sb
 then obtain a ∉ L a ∉ all-unshared sb
  by auto
 then show ?thesis by (clarsimp simp add: Ghost_sb True)
      qed
    next
      case False
      have dom (S ⊕_W R ⊖_A L) ⊆ dom (S ⊕_W (R − L))
        by auto
      from share-mono [OF this] a-in
      have a ∈ dom (share sb (S ⊕_W  (R − L))) by blast
      from share-union [OF this False]
      have a ∈ dom (share sb (Map.empty ⊕_W (R − L))).
      moreover
      have dom (Map.empty ⊕_W (R − L)) = dom (Map.empty ⊕_W R ⊖_A L)
        by auto
```

note dom-eq-dom-share-eq [OF this, of sb]
ultimately
show ?thesis
by (clarsimp simp add: $\mathrm{Ghost}_{\mathsf{sb}}$ False)
qed
qed
qed

lemma dom-augment-rels-shared-eq: dom (augment-rels S R $\mathcal{R}$) = dom (augment-rels S′ R $\mathcal{R}$)
by (auto simp add: augment-rels-def domIff split: option.splits if-split-asm)

lemma dom-eq-SomeD1: dom m = dom n $\Longrightarrow$ m x = Some y $\Longrightarrow$ n x $\neq$ None
by (auto simp add: dom-def)

lemma dom-eq-SomeD2: dom m = dom n $\Longrightarrow$ n x = Some y $\Longrightarrow$ m x $\neq$ None
by (auto simp add: dom-def)

lemma dom-augment-rels-rels-eq: dom $\mathcal{R}'$ = dom $\mathcal{R}$ $\Longrightarrow$ dom (augment-rels S R $\mathcal{R}'$) = dom (augment-rels S R $\mathcal{R}$)
by (auto simp add: augment-rels-def domIff split: option.splits if-split-asm dest: dom-eq-SomeD1 dom-eq-SomeD2)

lemma dom-release-rels-eq: $\bigwedge\mathcal{S}\ \mathcal{R}\ \mathcal{R}'$. dom $\mathcal{R}'$ = dom $\mathcal{R}$ $\Longrightarrow$
dom (release sb $\mathcal{S}$ $\mathcal{R}'$) = dom (release sb $\mathcal{S}$ $\mathcal{R}$)
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
hence dr: dom $\mathcal{R}'$ = dom $\mathcal{R}$
by simp
show ?case
proof (cases x)
case $\mathrm{Write}_{\mathsf{sb}}$ **with** Cons.hyps [OF dr] **show** ?thesis **by** (clarsimp)
next
case $\mathrm{Read}_{\mathsf{sb}}$ **with** Cons.hyps [OF dr] **show** ?thesis **by** (clarsimp)
next
case $\mathrm{Prog}_{\mathsf{sb}}$ **with** Cons.hyps [OF dr] **show** ?thesis **by** (clarsimp)
next
case ($\mathrm{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.hyps [OF dom-augment-rels-rels-eq [OF dr]]
show ?thesis
by (simp add: $\mathrm{Ghost}_{\mathsf{sb}}$)
qed
qed

lemma dom-release-shared-eq: $\bigwedge\mathcal{S}\ \mathcal{S}'\ \mathcal{R}$. dom (release sb $\mathcal{S}'$ $\mathcal{R}$) = dom (release sb $\mathcal{S}$ $\mathcal{R}$)
proof (induct sb)
  case Nil **thus** ?case **by** simp
next
  case (Cons x sb)
  show ?case
  proof (cases x)
    case $\text{Write}_{\mathsf{sb}}$ **with** Cons.hyps **show** ?thesis **by** (clarsimp)
  next
    case $\text{Read}_{\mathsf{sb}}$ **with** Cons.hyps **show** ?thesis **by** (clarsimp)
  next
    case $\text{Prog}_{\mathsf{sb}}$ **with** Cons.hyps **show** ?thesis **by** (clarsimp)
  next
    case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
    have dr: dom (augment-rels $\mathcal{S}'$ R $\mathcal{R}$) = dom (augment-rels $\mathcal{S}$ R $\mathcal{R}$)
      by(rule dom-augment-rels-shared-eq)
    have dom (release sb ($\mathcal{S}' \cup$ R $-$ L) (augment-rels $\mathcal{S}'$ R $\mathcal{R}$)) =
          dom (release sb ($\mathcal{S} \cup$ R $-$ L) (augment-rels $\mathcal{S}'$ R $\mathcal{R}$))
      by (rule Cons.hyps)
    also have ... = dom (release sb ($\mathcal{S} \cup$ R $-$ L) (augment-rels $\mathcal{S}$ R $\mathcal{R}$))
      by (rule dom-release-rels-eq [OF dr])
    finally show ?thesis
      by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
  qed
qed

lemma share-other-untouched:
  $\bigwedge\mathcal{O}\ \mathcal{S}$. sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb $\Longrightarrow$ a $\notin \mathcal{O} \cup$ all-acquired sb$\Longrightarrow$ share sb $\mathcal{S}$ a = $\mathcal{S}$ a
proof (induct sb)
  case Nil **thus** ?case **by** simp
next
  case (Cons x sb)
  show ?case
  proof (cases x)
    case ($\text{Write}_{\mathsf{sb}}$ volatile a' sop v A L R W)
    show ?thesis
    proof (cases volatile)
      case True

      from Cons.prems **obtain**
 A-shared-owns: A $\subseteq$ dom $\mathcal{S} \cup \mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq \mathcal{O}$ **and**
 consis': sharing-consistent ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb **and**
      a-owns: a $\notin \mathcal{O}$ **and** a-A: a $\notin$ A **and** a-sb: a $\notin$ all-acquired sb
 by (simp add: $\text{Write}_{\mathsf{sb}}$ True)

      from a-owns a-A a-sb
      have a $\notin \mathcal{O} \cup$ A $-$ R $\cup$ all-acquired sb
        by auto

from Cons.hyps [OF consis′ this]
have share sb ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a.
moreover have ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = $\mathcal{S}$ a
using L-A A-R R-owns a-owns a-A
by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
ultimately show ?thesis
by (simp add: $\text{Write}_{\mathsf{sb}}$ True)
next
case False **with** Cons **show** ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ False)
qed
next
case $\text{Read}_{\mathsf{sb}}$ **with** Cons
show ?thesis
by (auto)
next
case $\text{Prog}_{\mathsf{sb}}$ **with** Cons
show ?thesis
by (auto)
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.prems **obtain**
A-shared-owns: A $\subseteq$ dom $\mathcal{S} \cup \mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq \mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb **and**
a-owns: a $\notin \mathcal{O}$ **and** a-A: a $\notin$ A **and** a-sb: a $\notin$ all-acquired sb
by (simp add: $\text{Ghost}_{\mathsf{sb}}$)

from a-owns a-A a-sb
have a $\notin \mathcal{O} \cup$ A $-$ R $\cup$ all-acquired sb
by auto
from Cons.hyps [OF consis′ this]
have share sb ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a.
moreover have ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = $\mathcal{S}$ a
using L-A A-R R-owns a-owns a-A
by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
ultimately show ?thesis
by (simp add: $\text{Ghost}_{\mathsf{sb}}$)
qed
qed

lemma shared-owned: $\bigwedge\mathcal{O}\ \mathcal{S}$. sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb $\Longrightarrow$ a $\notin$ dom $\mathcal{S}$ $\Longrightarrow$ a $\in$ dom (share sb $\mathcal{S}$) $\Longrightarrow$
a $\in \mathcal{O} \cup$ all-acquired sb
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)

case (Write$_{\mathsf{sb}}$ volatile a$'$ sop v A L R W)
show ?thesis
proof (cases volatile)
case True

from Cons.prems **obtain**
A-shared-owns: A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and**
consis$'$: sharing-consistent ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ $\cup$ A $-$ R) sb **and**
a-notin: a $\notin$ dom $\mathcal{S}$ **and** a-in: a $\in$ dom (share sb ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L))
by (simp add: Write$_{\mathsf{sb}}$ True)

show ?thesis
proof (cases a $\in$ $\mathcal{O}$)
case True **thus** ?thesis **by** auto
next
case False
with a-notin R-owns A-shared-owns L-A A-R **have** a $\notin$ dom ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (auto)
from Cons.hyps [OF consis$'$ this a-in]
show ?thesis
by (auto simp add: Write$_{\mathsf{sb}}$ True)
qed
next
case False **with** Cons **show** ?thesis
by (auto simp add: Write$_{\mathsf{sb}}$ False)
qed
next
case Read$_{\mathsf{sb}}$ **with** Cons
show ?thesis
by (auto)
next
case Prog$_{\mathsf{sb}}$ **with** Cons
show ?thesis
by (auto)
next
case (Ghost$_{\mathsf{sb}}$ A L R W)
from Cons.prems **obtain**
A-shared-owns: A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and**
consis$'$: sharing-consistent ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ $\cup$ A $-$ R) sb **and**
a-notin: a $\notin$ dom $\mathcal{S}$ **and** a-in: a $\in$ dom (share sb ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L))
by (simp add: Ghost$_{\mathsf{sb}}$)

show ?thesis
proof (cases a $\in$ $\mathcal{O}$)
case True **thus** ?thesis **by** auto
next
case False
with a-notin R-owns A-shared-owns L-A A-R **have** a $\notin$ dom ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)

by (auto)
from Cons.hyps [OF consis$'$ this a-in]
show ?thesis
by (auto simp add: $\mathsf{Ghost_{sb}}$)
qed
qed
qed

lemma share-all-shared-in: a $\in$ dom (share sb $\mathcal{S}$) $\Longrightarrow$ a $\in$ dom $\mathcal{S}$ $\vee$ a $\in$ all-shared sb
using sharing-consistent-share-all-shared [of sb $\mathcal{S}$]
by auto

lemma share-all-until-volatile-write-unowned:
assumes dist: ownership-distinct ts
assumes consis: sharing-consis $\mathcal{S}$ ts
assumes other: $\forall$i p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$. i $<$ length ts $\longrightarrow$ ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
a $\notin$ $\mathcal{O}$ $\cup$ all-acquired sb
shows share-all-until-volatile-write ts $\mathcal{S}$ a = $\mathcal{S}$ a
using dist consis other
proof (induct ts arbitrary: $\mathcal{S}$)
case Nil **thus** ?case **by** simp
next
case (Cons t ts)
obtain $\mathrm{p_t}$ $\mathrm{is_t}$ $\mathcal{O}_\mathsf{t}$ $\mathcal{R}_\mathsf{t}$ $\mathcal{D}_\mathsf{t}$ $\mathrm{j_t}$ $\mathrm{sb_t}$ **where**
t: t=($\mathrm{p_t}$,$\mathrm{is_t}$,$\mathrm{j_t}$,$\mathrm{sb_t}$,$\mathcal{D}_\mathsf{t}$,$\mathcal{O}_\mathsf{t}$,$\mathcal{R}_\mathsf{t}$)
by (cases t)

from Cons.prems t **obtain**
other$'$: $\forall$i p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$. i $<$ length ts $\longrightarrow$ ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
a $\notin$ $\mathcal{O}$ $\cup$ all-acquired sb **and**
a-notin: a $\notin$ $\mathcal{O}_\mathsf{t}$ $\cup$ all-acquired $\mathrm{sb_t}$
apply $-$
apply (rule that)
apply clarsimp
subgoal for i p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$
apply (drule-tac x=Suc i **in** spec)
apply clarsimp
done
apply (drule-tac x=0 **in** spec)
apply clarsimp
done

have dist: ownership-distinct (t#ts) **by** fact
then interpret ownership-distinct t#ts**.**
have consis: sharing-consis $\mathcal{S}$ (t#ts) **by** fact
then interpret sharing-consis $\mathcal{S}$ t#ts**.**

from ownership-distinct-tl [OF dist]
have dist$'$: ownership-distinct ts**.**

from sharing-consis-tl [OF consis]
have consis$'$: sharing-consis $\mathcal{S}$ ts**.**
then
interpret consis$'$: sharing-consis $\mathcal{S}$ ts**.**

let ?$\mathcal{S}'$ = (share (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_{\mathsf{t}}$) $\mathcal{S}$)

from sharing-consis [of 0, simplified, OF t]
have sharing-consistent $\mathcal{S}$ $\mathcal{O}_{\mathsf{t}}$ sb$_{\mathsf{t}}$**.**
from sharing-consistent-takeWhile [OF this]
have consis-sb: sharing-consistent $\mathcal{S}$ $\mathcal{O}_{\mathsf{t}}$ (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_{\mathsf{t}}$)**.**
from freshly-shared-owned [OF consis-sb]
have fresh-owned: dom ?$\mathcal{S}'$ $-$ dom $\mathcal{S}$ $\subseteq$ $\mathcal{O}_{\mathsf{t}}$**.**
from unshared-all-unshared [OF consis-sb] unshared-acquired-or-owned [OF consis-sb]
have unshared-acq-owned: dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$
$\subseteq$ all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_{\mathsf{t}}$) $\cup$ $\mathcal{O}_{\mathsf{t}}$
by simp

have sep:
$\forall$i < length ts. let (-,-,-,sb$'$,-,-,-) = ts!i in
all-acquired sb$'$ $\cap$ dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$ = {} $\wedge$
all-unshared sb$'$ $\cap$ dom ?$\mathcal{S}'$ $-$ dom $\mathcal{S}$ = {}
proof $-$
{
fix i p$_{\mathsf{i}}$ is$_{\mathsf{i}}$ $\mathcal{O}_{\mathsf{i}}$ $\mathcal{R}_{\mathsf{i}}$ $\mathcal{D}_{\mathsf{i}}$ j$_{\mathsf{i}}$ sb$_{\mathsf{i}}$
assume i-bound: i < length ts
assume ts-i: ts ! i = (p$_{\mathsf{i}}$,is$_{\mathsf{i}}$,j$_{\mathsf{i}}$,sb$_{\mathsf{i}}$,$\mathcal{D}_{\mathsf{i}}$,$\mathcal{O}_{\mathsf{i}}$,$\mathcal{R}_{\mathsf{i}}$)
have all-acquired sb$_{\mathsf{i}}$ $\cap$ dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$ = {} $\wedge$
all-unshared sb$_{\mathsf{i}}$ $\cap$ dom ?$\mathcal{S}'$ $-$ dom $\mathcal{S}$ = {}
proof $-$
from ownership-distinct [of 0 Suc i] ts-i t i-bound
have dist: ($\mathcal{O}_{\mathsf{t}}$ $\cup$ all-acquired sb$_{\mathsf{t}}$) $\cap$ ($\mathcal{O}_{\mathsf{i}}$ $\cup$ all-acquired sb$_{\mathsf{i}}$) = {}
by force

from dist unshared-acq-owned all-acquired-takeWhile [of (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_{\mathsf{t}}$]
have all-acquired sb$_{\mathsf{i}}$ $\cap$ dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$ = {}
by blast

moreover

from sharing-consis [of Suc i] ts-i i-bound
have sharing-consistent $\mathcal{S}$ $\mathcal{O}_{\mathsf{i}}$ sb$_{\mathsf{i}}$
by force
from unshared-acquired-or-owned [OF this]
have all-unshared sb$_{\mathsf{i}}$ $\subseteq$ all-acquired sb$_{\mathsf{i}}$ $\cup$ $\mathcal{O}_{\mathsf{i}}$**.**
with dist fresh-owned

have all-unshared sb_i ∩ dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ = {}
by blast

ultimately show ?thesis **by** simp
qed
}
thus ?thesis
by (fastforce simp add: Let-def)
qed

from consis′.sharing-consis-preservation [OF this]
have sharing-consis ?$\mathcal{S}'$ ts.

from Cons.hyps [OF dist′ this other′]
have share-all-until-volatile-write ts ?$\mathcal{S}'$ a =
share (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_t) $\mathcal{S}$ a .
moreover
from share-other-untouched [OF consis-sb] a-notin
all-acquired-append [of (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_t) (dropWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_t)]
have share (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_t) $\mathcal{S}$ a = $\mathcal{S}$ a
by auto
ultimately
show ?case
by (simp add: t)
qed

lemma share-shared-eq: $\bigwedge\mathcal{S}'\ \mathcal{S}.\ \mathcal{S}'$ a = $\mathcal{S}$ a $\Longrightarrow$ share sb $\mathcal{S}'$ a = share sb $\mathcal{S}$ a
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
have eq: $\mathcal{S}'$ a = $\mathcal{S}$ a **by** fact
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a′ sop v A L R W)
show ?thesis
proof (cases volatile)
case True

have ($\mathcal{S}'$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a
using eq **by** (auto simp add: augment-shared-def restrict-shared-def)
from Cons.hyps [of ($\mathcal{S}'$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L), OF this]
show ?thesis
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ True)
next
case False
with Cons.hyps [of $\mathcal{S}'$ $\mathcal{S}$] Cons.prems **show** ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ False)
qed

next
  case $\text{Read}_{\mathsf{sb}}$
  with Cons.hyps [of $\mathcal{S}'$ $\mathcal{S}$] Cons.prems **show** ?thesis
    by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
next
  case $\text{Prog}_{\mathsf{sb}}$
  with Cons.hyps [of $\mathcal{S}'$ $\mathcal{S}$] Cons.prems **show** ?thesis
    by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
next
  case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
  have ($\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a
  using eq **by** (auto simp add: augment-shared-def restrict-shared-def)
  from Cons.hyps [of ($\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L), OF this]
  show ?thesis
    by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
qed
qed

lemma share-all-until-volatile-write-thread-local:
  assumes dist: ownership-distinct ts
  assumes consis: sharing-consis $\mathcal{S}$ ts
  assumes i-bound: i < length ts
  assumes ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  assumes a-owned: a $\in \mathcal{O} \cup$ all-acquired sb
  shows share-all-until-volatile-write ts $\mathcal{S}$ a = share (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) $\mathcal{S}$ a
using dist consis i-bound ts-i
proof (induct ts arbitrary: $\mathcal{S}$ i)
  case Nil **thus** ?case **by** simp
next
  case (Cons t ts)

  obtain p_{t} is_{t} $\mathcal{O}_{\mathsf{t}}$ $\mathcal{R}_{\mathsf{t}}$ $\mathcal{D}_{\mathsf{t}}$ j_{t} sb_{t} **where**
    t: t=(p_{t},is_{t},j_{t},sb_{t},$\mathcal{D}_{\mathsf{t}}$,$\mathcal{O}_{\mathsf{t}}$,$\mathcal{R}_{\mathsf{t}}$)
    by (cases t)

  have dist: ownership-distinct (t#ts) **by** fact
  then interpret ownership-distinct t#ts**.**
  have consis: sharing-consis $\mathcal{S}$ (t#ts) **by** fact
  then interpret sharing-consis $\mathcal{S}$ t#ts**.**

  from ownership-distinct-tl [OF dist]
  have dist′: ownership-distinct ts**.**

  from sharing-consis-tl [OF consis]
  have consis′: sharing-consis $\mathcal{S}$ ts**.**
  then
  interpret consis′: sharing-consis $\mathcal{S}$ ts**.**
  let ?$\mathcal{S}'$ = (share (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{t}) $\mathcal{S}$)

from sharing-consis [of 0, simplified, OF t]
have sharing-consistent $\mathcal{S}$ $\mathcal{O}_{\mathsf{t}}$ sb_{t}.
from sharing-consistent-takeWhile [OF this]
have consis-sb: sharing-consistent $\mathcal{S}$ $\mathcal{O}_{\mathsf{t}}$ (takeWhile (Not ∘ is-volatile-$\mathrm{Write}_{\mathsf{sb}}$) sb_{t}).
from freshly-shared-owned [OF consis-sb]
have fresh-owned: dom ?$\mathcal{S}'$ − dom $\mathcal{S} \subseteq \mathcal{O}_{\mathsf{t}}$.
from unshared-all-unshared [OF consis-sb] unshared-acquired-or-owned [OF consis-sb]
have unshared-acq-owned: dom $\mathcal{S}$ − dom ?$\mathcal{S}'$
$\subseteq$ all-acquired (takeWhile (Not ∘ is-volatile-$\mathrm{Write}_{\mathsf{sb}}$) sb_{t}) ∪ $\mathcal{O}_{\mathsf{t}}$
by simp

have sep:
∀ i < length ts. let (-,-,-,sb′,-,-,-) = ts!i in
all-acquired sb′ ∩ dom $\mathcal{S}$ − dom ?$\mathcal{S}'$ = {} ∧
all-unshared sb′ ∩ dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ = {}
proof −
{
fix i p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j_i sb_i
assume i-bound: i < length ts
assume ts-i: ts ! i = (p_i,is_i,j_i,sb_i,$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$)
have all-acquired sb_i ∩ dom $\mathcal{S}$ − dom ?$\mathcal{S}'$ = {} ∧
all-unshared sb_i ∩ dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ = {}
proof −
from ownership-distinct [of 0 Suc i] ts-i t i-bound
have dist: ($\mathcal{O}_{\mathsf{t}}$ ∪ all-acquired sb_{t}) ∩ ($\mathcal{O}_i$ ∪ all-acquired sb_i) = {}
by force

from dist unshared-acq-owned all-acquired-takeWhile [of (Not ∘ is-volatile-$\mathrm{Write}_{\mathsf{sb}}$) sb_{t}]
have all-acquired sb_i ∩ dom $\mathcal{S}$ − dom ?$\mathcal{S}'$ = {}
by blast

moreover

from sharing-consis [of Suc i] ts-i i-bound
have sharing-consistent $\mathcal{S}$ $\mathcal{O}_i$ sb_i
by force
from unshared-acquired-or-owned [OF this]
have all-unshared sb_i ⊆ all-acquired sb_i ∪ $\mathcal{O}_i$.
with dist fresh-owned
have all-unshared sb_i ∩ dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ = {}
by blast

ultimately show ?thesis **by** simp
qed
}
thus ?thesis
by (fastforce simp add: Let-def)
qed

from consis′.sharing-consis-preservation [OF this]
have consis-shared′: sharing-consis ?$\mathcal{S}'$ ts**.**

have aargh: (Not ∘ is-volatile-Write$_{sb}$) = (λa. ¬ is-volatile-Write$_{sb}$ a)
  by (rule ext) auto

show ?case
proof (cases i)
  case 0
  with Cons.prems
  have t′: t = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$)
    by simp

  {
    fix j p_j is_j j_j sb_j $\mathcal{D}_j$ $\mathcal{O}_j$ $\mathcal{R}_j$
    assume j-bound: j < length ts
    assume ts-j: ts ! j = (p_j, is_j, j_j, sb_j, $\mathcal{D}_j$, $\mathcal{O}_j$, $\mathcal{R}_j$)
    have a ∉ $\mathcal{O}_j$ ∪ all-acquired sb_j
    proof −
      from ownership-distinct [of 0 Suc j, simplified, OF j-bound t ts-j] t a-owned t′ 0
      show ?thesis
        by auto
    qed
  }

  with share-all-until-volatile-write-unowned [OF dist′ consis-shared′, of a]
  have share-all-until-volatile-write ts ?$\mathcal{S}'$ a = ?$\mathcal{S}'$ a
    by fastforce
  then show ?thesis
  using t t′ 0
    by (auto simp add: Cons t aargh)
next
  case (Suc n)
  with Cons.prems **obtain** n-bound: n < length ts **and** ts-n: ts!n = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
    by auto
  from Cons.hyps [OF dist′ consis-shared′ n-bound ts-n]
  have share-all-until-volatile-write ts ?$\mathcal{S}'$ a =
        share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ?$\mathcal{S}'$ a **.**
  moreover
  from ownership-distinct [of 0 Suc n] t a-owned ts-n n-bound
  have a ∉ $\mathcal{O}_t$ ∪ all-acquired sb_t
    by fastforce
  with share-other-untouched [OF consis-sb, of a]
   all-acquired-append [of (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_t) (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb_t)]
  have ?$\mathcal{S}'$ a = $\mathcal{S}$ a
    by auto
  from share-shared-eq [of ?$\mathcal{S}'$ a $\mathcal{S}$,OF this]

have share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ?$\mathcal{S}'$ a =
share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{S}$ a **.**
ultimately show ?thesis
using t Suc
by (auto simp add: aargh)
qed
qed

lemma share-all-until-volatile-write-thread-local$'$:
assumes dist: ownership-distinct ts
assumes consis: sharing-consis $\mathcal{S}$ ts
assumes i-bound: i < length ts
assumes ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
assumes a-owned: a ∈ $\mathcal{O}$ ∪ all-acquired sb
shows share (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb) (share-all-until-volatile-write ts $\mathcal{S}$) a =
share sb $\mathcal{S}$ a
proof −
let ?take = takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb
let ?drop = dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb
from share-all-until-volatile-write-thread-local [OF dist consis i-bound ts-i a-owned]
have share-all-until-volatile-write ts $\mathcal{S}$ a = share ?take $\mathcal{S}$ a **.**
moreover
from share-shared-eq [of share-all-until-volatile-write ts $\mathcal{S}$ a share ?take $\mathcal{S}$, OF this]
have share ?drop (share-all-until-volatile-write ts $\mathcal{S}$) a = share ?drop (share ?take $\mathcal{S}$) a **.**
thus ?thesis
using share-append [of ?take ?drop $\mathcal{S}$]
by simp
qed

lemma (**in** ownership-distinct) in-shared-sb-share-all-until-volatile-write:
assumes consis: sharing-consis $\mathcal{S}$ ts
assumes i-bound: i < length ts
assumes ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
assumes a-owned: a ∈ $\mathcal{O}$ ∪ all-acquired sb
assumes a-share: a ∈ dom (share sb $\mathcal{S}$)
shows a ∈ dom (share (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
(share-all-until-volatile-write ts $\mathcal{S}$))
proof −
have dist: ownership-distinct ts
using assms ownership-distinct
apply −
apply (rule ownership-distinct.intro)
apply auto
done
from share-all-until-volatile-write-thread-local$'$ [OF dist consis i-bound ts-i a-owned]
a-share
show ?thesis
by (auto simp add: domIff)
qed

lemma owns-unshared-share-acquired:
$\bigwedge\mathcal{S}\ \mathcal{O}$. $[\![$sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb; a $\in \mathcal{O}$; a $\notin$ all-unshared sb$]\!]$
$\Longrightarrow$ a $\in$ dom (share sb $\mathcal{S}$) $\cup$ acquired True sb $\mathcal{O}$
proof (induct sb)
case Nil **thus** ?case **by** auto
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a$'$ sop v A L R W)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems **obtain**
a-owns: a $\in \mathcal{O}$ **and** A-shared-onws: A $\subseteq$ dom $\mathcal{S} \cup \mathcal{O}$ **and**
a-L: a $\notin$ L **and** a-unsh: a $\notin$ all-unshared sb **and** L-A: L $\subseteq$ A **and**
A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq \mathcal{O}$ **and**
consis$'$: sharing-consistent ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ volatile)
have a $\in$ dom (share sb ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) $\cup$ acquired True sb ($\mathcal{O} \cup$ A $-$ R)
proof (cases a $\in$ R)
case True
with a-L **have** a $\in$ dom ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by auto
from unshared-share-in [OF this a-unsh]
show ?thesis **by** blast
next
case False
hence a $\in \mathcal{O} \cup$ A $-$ R
using a-owns
by auto
from Cons.hyps [OF consis$'$ this a-unsh]
show ?thesis **.**
qed
then
show ?thesis
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ volatile)
next
case False
with Cons
show ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$)
qed
next
case $\text{Read}_{\mathsf{sb}}$
with Cons **show** ?thesis
by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
next

case Prog$_{sb}$
with Cons show ?thesis
by (auto simp add: Read$_{sb}$)
next
case (Ghost$_{sb}$ A L R W)
from Cons.prems obtain
a-owns: $a \in \mathcal{O}$ and A-shared-onws: $A \subseteq \text{dom } \mathcal{S} \cup \mathcal{O}$ and
a-L: $a \notin L$ and a-unsh: $a \notin$ all-unshared sb and L-A: $L \subseteq A$ and
A-R: $A \cap R = \{\}$ and R-owns: $R \subseteq \mathcal{O}$ and
consis$'$: sharing-consistent $(\mathcal{S} \oplus_W R \ominus_A L)$ $(\mathcal{O} \cup A - R)$ sb
by (clarsimp simp add: Ghost$_{sb}$)
have $a \in$ dom (share sb $(\mathcal{S} \oplus_W R \ominus_A L)$) $\cup$ acquired True sb $(\mathcal{O} \cup A - R)$
proof (cases $a \in R$)
case True
with a-L have $a \in \text{dom } (\mathcal{S} \oplus_W R \ominus_A L)$
by auto
from unshared-share-in [OF this a-unsh]
show ?thesis by blast
next
case False
hence $a \in \mathcal{O} \cup A - R$
using a-owns
by auto
from Cons.hyps [OF consis$'$ this a-unsh]
show ?thesis .
qed
then show ?thesis
by (auto simp add: Ghost$_{sb}$)
qed
qed

lemma shared-share-acquired: $\bigwedge \mathcal{S}\ \mathcal{O}$. sharing-consistent $\mathcal{S}\ \mathcal{O}$ sb $\Longrightarrow$
$a \in \text{dom } \mathcal{S} \Longrightarrow a \in$ dom (share sb $\mathcal{S}$) $\cup$ acquired True sb $\mathcal{O}$
proof (induct sb)
case Nil thus ?case by auto
next
case (Cons x sb)
show ?case
proof (cases x)
case (Write$_{sb}$ volatile a$'$ sop v A L R W)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems obtain
a-shared: $a \in \text{dom } \mathcal{S}$ and A-shared-owns: $A \subseteq \text{dom } \mathcal{S} \cup \mathcal{O}$ and
L-A: $L \subseteq A$ and A-R: $A \cap R = \{\}$ and R-owns: $R \subseteq \mathcal{O}$ and
consis$'$: sharing-consistent $(\mathcal{S} \oplus_W R \ominus_A L)$ $(\mathcal{O} \cup A - R)$ sb
by (clarsimp simp add: Write$_{sb}$ True)
show ?thesis

proof (cases $a \in L$)
case False **with** a-shared
have $a \in$ dom ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by auto
from Cons.hyps [OF consis′ this]
show ?thesis
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ volatile)
next
case True
with L-A **have** a-A: $a \in A$
by blast
from sharing-consistent-mono-shared [OF - consis′, **where** $\mathcal{S}'$=($\mathcal{S} \oplus_{\mathsf{W}}$ R)]
have sharing-consistent ($\mathcal{S} \oplus_{\mathsf{W}}$ R) ($\mathcal{O} \cup$ A $-$ R) sb
by auto
from Cons.hyps [OF this] a-shared
have hyp: $a \in$ dom (share sb ($\mathcal{S} \oplus_{\mathsf{W}}$ R)) $\cup$ acquired True sb ($\mathcal{O} \cup$ A $-$ R)
by auto
{
assume $a \in$ dom (share sb ($\mathcal{S} \oplus_{\mathsf{W}}$ R))
from share-unshared-in [OF this]
have $a \in$ dom (share sb ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) $\cup$ acquired True sb ($\mathcal{O} \cup$ A $-$ R)
proof
assume $a \in$ dom (share sb Map.empty)
from share-mono-in [OF this]
have $a \in$ dom (share sb ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L))
by auto
thus ?thesis **by** blast
next
assume $a \in$ dom ($\mathcal{S} \oplus_{\mathsf{W}}$ R) $\wedge$ $a \notin$ all-unshared sb
hence a-unsh: $a \notin$ all-unshared sb **by** blast
from a-A A-R **have** $a \in \mathcal{O} \cup$ A $-$ R
by auto
from owns-unshared-share-acquired [OF consis′ this a-unsh]
show ?thesis **.**
qed
}
with hyp **show** ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ volatile)
qed
next
case False
with Cons
show ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$)
qed
next
case $\text{Read}_{\mathsf{sb}}$
with Cons **show** ?thesis
by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
next

```
    case Prog_sb
    with Cons show ?thesis
      by (auto simp add: Read_sb)
  next
    case (Ghost_sb A L R W)
    from Cons.prems obtain
      a-shared: a ∈ dom 𝒮 and A-shared-owns: A ⊆ dom 𝒮 ∪ 𝒪 and
      L-A: L ⊆ A and A-R: A ∩ R = {} and R-owns: R ⊆ 𝒪 and
      consis': sharing-consistent (𝒮 ⊕_W R ⊖_A L) (𝒪 ∪ A − R) sb
      by (clarsimp simp add: Ghost_sb)
    show ?thesis
    proof (cases a ∈ L)
      case False with a-shared
      have a ∈ dom (𝒮 ⊕_W R ⊖_A L)
        by auto
      from Cons.hyps [OF consis' this]
      show ?thesis
        by (clarsimp simp add: Ghost_sb)
    next
      case True
      with L-A have a-A: a ∈ A
        by blast
      from sharing-consistent-mono-shared [OF - consis', where 𝒮'=(𝒮 ⊕_W R)]
      have sharing-consistent (𝒮 ⊕_W R) (𝒪 ∪ A − R) sb
        by auto
      from Cons.hyps [OF this] a-shared
      have hyp: a ∈ dom (share sb (𝒮 ⊕_W R)) ∪ acquired True sb (𝒪 ∪ A − R)
        by auto
      {
 assume a ∈ dom (share sb (𝒮 ⊕_W R))
 from share-unshared-in [OF this]
 have a ∈ dom (share sb (𝒮 ⊕_W R ⊖_A L)) ∪ acquired True sb (𝒪 ∪ A − R)
        proof
   assume a ∈ dom (share sb Map.empty)
   from share-mono-in [OF this]
   have a ∈ dom (share sb (𝒮 ⊕_W R ⊖_A L))
     by auto
   thus ?thesis by blast
        next
   assume a ∈ dom (𝒮 ⊕_W R) ∧ a ∉ all-unshared sb
   hence a-unsh: a ∉ all-unshared sb by blast
   from a-A A-R have a ∈ 𝒪 ∪ A − R
     by auto
   from owns-unshared-share-acquired [OF consis' this a-unsh]
   show ?thesis .
        qed
      }
      with hyp show ?thesis
        by (auto simp add: Ghost_sb)
    qed
```

qed
qed

lemma dom-release-takeWhile:
$\bigwedge$S $\mathcal{R}$.
dom (release (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) S $\mathcal{R}$) =
dom $\mathcal{R}$ ∪ all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
apply (induct sb)
apply (clarsimp)
subgoal for a sb S $\mathcal{R}$
apply (case-tac a)
apply (auto simp add: augment-rels-def domIff split: if-split-asm option.splits)
done
done

lemma share-all-until-volatile-write-share-acquired:
assumes dist: ownership-distinct ts
assumes consis: sharing-consis $\mathcal{S}$ ts
assumes a-notin: a ∉ dom $\mathcal{S}$
assumes a-in: a ∈ dom (share-all-until-volatile-write ts $\mathcal{S}$)
shows ∃ i < length ts.
let (-,-,-,sb,-,-,-) = ts!i
in a ∈ all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
using dist consis a-notin a-in
proof (induct ts arbitrary: $\mathcal{S}$ i)
case Nil **thus** ?case **by** simp
next
case (Cons t ts)

have a-notin: a ∉ dom $\mathcal{S}$ **by** fact
obtain p_t is_t $\mathcal{O}_t$ $\mathcal{R}_t$ $\mathcal{D}_t$ j_t sb_t **where**
t: t=(p_t,is_t,j_t,sb_t,$\mathcal{D}_t$,$\mathcal{O}_t$,$\mathcal{R}_t$)
by (cases t)

let ?take = (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_t)
from t Cons.prems
have a-in: a ∈ dom (share-all-until-volatile-write ts (share ?take $\mathcal{S}$))
by auto

have dist: ownership-distinct (t#ts) **by** fact
then interpret ownership-distinct t#ts**.**
have consis: sharing-consis $\mathcal{S}$ (t#ts) **by** fact
then interpret sharing-consis $\mathcal{S}$ t#ts**.**

from ownership-distinct-tl [OF dist]
have dist′: ownership-distinct ts**.**

from sharing-consis-tl [OF consis]
have consis′: sharing-consis $\mathcal{S}$ ts**.**
then

interpret consis′: sharing-consis $\mathcal{S}$ ts.
let ?$\mathcal{S}'$ = (share ?take $\mathcal{S}$)

from sharing-consis [of 0, simplified, OF t]
have sharing-consistent $\mathcal{S}$ $\mathcal{O}_\mathsf{t}$ sb_t.
from sharing-consistent-takeWhile [OF this]
have consis-sb: sharing-consistent $\mathcal{S}$ $\mathcal{O}_\mathsf{t}$?take.
from freshly-shared-owned [OF consis-sb]
have fresh-owned: dom ?$\mathcal{S}'$ − dom $\mathcal{S} \subseteq \mathcal{O}_\mathsf{t}$.
from unshared-all-unshared [OF consis-sb] unshared-acquired-or-owned [OF consis-sb]
have unshared-acq-owned: dom $\mathcal{S}$ − dom ?$\mathcal{S}'$
$\subseteq$ all-acquired ?take $\cup$ $\mathcal{O}_\mathsf{t}$
by simp

have sep:
$\forall$i < length ts. let (-,-,-,sb′,-,-,-) = ts!i in
all-acquired sb′ $\cap$ dom $\mathcal{S}$ − dom ?$\mathcal{S}'$ = {} $\wedge$
all-unshared sb′ $\cap$ dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ = {}
proof −
{
fix i $\mathrm{p_i}$ $\mathrm{is_i}$ $\mathcal{O}_\mathsf{i}$ $\mathcal{R}_\mathsf{i}$ $\mathcal{D}_\mathsf{i}$ $\mathrm{j_i}$ $\mathrm{sb_i}$
assume i-bound: i < length ts
assume ts-i: ts ! i = ($\mathrm{p_i}$,$\mathrm{is_i}$,$\mathrm{j_i}$,$\mathrm{sb_i}$,$\mathcal{D}_\mathsf{i}$,$\mathcal{O}_\mathsf{i}$,$\mathcal{R}_\mathsf{i}$)
have all-acquired $\mathrm{sb_i}$ $\cap$ dom $\mathcal{S}$ − dom ?$\mathcal{S}'$ = {} $\wedge$
all-unshared $\mathrm{sb_i}$ $\cap$ dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ = {}
proof −
from ownership-distinct [of 0 Suc i] ts-i t i-bound
have dist: ($\mathcal{O}_\mathsf{t}$ $\cup$ all-acquired $\mathrm{sb_t}$) $\cap$ ($\mathcal{O}_\mathsf{i}$ $\cup$ all-acquired $\mathrm{sb_i}$) = {}
by force

from dist unshared-acq-owned all-acquired-takeWhile [of (Not $\circ$ is-volatile-$\mathrm{Write_{sb}}$) $\mathrm{sb_t}$]
have all-acquired $\mathrm{sb_i}$ $\cap$ dom $\mathcal{S}$ − dom ?$\mathcal{S}'$ = {}
by blast

moreover

from sharing-consis [of Suc i] ts-i i-bound
have sharing-consistent $\mathcal{S}$ $\mathcal{O}_\mathsf{i}$ $\mathrm{sb_i}$
by force
from unshared-acquired-or-owned [OF this]
have all-unshared $\mathrm{sb_i}$ $\subseteq$ all-acquired $\mathrm{sb_i}$ $\cup$ $\mathcal{O}_\mathsf{i}$.
with dist fresh-owned
have all-unshared $\mathrm{sb_i}$ $\cap$ dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ = {}
by blast

ultimately show ?thesis **by** simp
qed
}
thus ?thesis

```
      by (fastforce simp add: Let-def)
  qed

  from consis'.sharing-consis-preservation [OF this]
  have consis-shared': sharing-consis ?𝒮' ts.


  have aargh: (Not ∘ is-volatile-Write_sb) = (λa. ¬ is-volatile-Write_sb a)
    by (rule ext) auto

  show ?case
  proof (cases a ∈ all-shared ?take)
    case True
    thus ?thesis
    apply −
    apply (rule-tac x=0 in exI)
    apply (auto simp add: t aargh)
    done
  next
    case False

    have a-notin': a ∉ dom ?𝒮'
    proof
      assume a ∈ dom ?𝒮'
      from share-all-shared-in [OF this] False a-notin
      show False
        by auto
    qed
    from Cons.hyps [OF dist' consis-shared' a-notin' a-in]
    obtain i where i < length ts and
      rel: let (p,is,j,sb,𝒟,𝒪,ℛ) = ts!i
            in a ∈ all-shared (takeWhile (Not ∘ is-volatile-Write_sb) sb)
      by (auto simp add: Let-def aargh)
    then show ?thesis
      apply −
      apply (rule-tac x = Suc i in exI)
      apply (auto simp add: Let-def aargh)
      done
  qed
qed

lemma all-shared-share-acquired: ⋀𝒮 𝒪. sharing-consistent 𝒮 𝒪 sb ⟹
  a ∈ all-shared sb ⟹ a ∈ dom (share sb 𝒮) ∪ acquired True sb 𝒪
proof (induct sb)
  case Nil thus ?case by auto
next
  case (Cons x sb)
  show ?case
  proof (cases x)
    case (Write_sb volatile a' sop v A L R W)
```

show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems **obtain**
a-shared: a ∈ R ∪ all-shared sb **and** A-shared-owns: A ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$ **and**
L-A: L ⊆ A **and** A-R: A ∩ R = {} **and** R-owns: R ⊆ $\mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ ∪ A − R) sb
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ True)
show ?thesis
proof (cases a ∈ all-shared sb)
case True
from Cons.hyps [OF consis′ True]
show ?thesis
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ volatile)
next
case False
with a-shared **have** a ∈ R
by auto
with L-A A-R R-owns **have** a ∈ dom ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by auto
from shared-share-acquired [OF consis′ this]
show ?thesis
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ volatile)
qed
next
case False
with Cons
show ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$)
qed
next
case $\text{Read}_{\mathsf{sb}}$
with Cons **show** ?thesis
by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
next
case $\text{Prog}_{\mathsf{sb}}$
with Cons **show** ?thesis
by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.prems **obtain**
a-shared: a ∈ R ∪ all-shared sb **and** A-shared-owns: A ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$ **and**
L-A: L ⊆ A **and** A-R: A ∩ R = {} **and** R-owns: R ⊆ $\mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ ∪ A − R) sb
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
show ?thesis
proof (cases a ∈ all-shared sb)
case True
from Cons.hyps [OF consis′ True]

show ?thesis
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
next
case False
with a-shared **have** a ∈ R
by auto
with L-A A-R R-owns **have** a ∈ dom ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by auto
from shared-share-acquired [OF consis′ this]
show ?thesis
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
qed
qed
qed

lemma (**in** ownership-distinct) share-all-until-volatile-write-share-acquired:
assumes consis: sharing-consis $\mathcal{S}$ ts
assumes i-bound: i < length ts
assumes ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
assumes a-in: a ∈ dom (share-all-until-volatile-write ts $\mathcal{S}$)
shows a ∈ dom (share sb $\mathcal{S}$) ∨ a ∈ acquired True sb $\mathcal{O}$ ∨
(∃ j < length ts. j ≠ i ∧
(let (-,-,-,sb_{j},-,-,-) = ts!j
in a ∈ all-shared (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{j})))
proof −
from assms ownership-distinct **have** dist: ownership-distinct ts
apply −
apply (rule ownership-distinct.intro)
apply simp
done
from consis
interpret sharing-consis $\mathcal{S}$ ts **.**
from sharing-consis [OF i-bound ts-i]
have consis-sb: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb**.**

let ?take-sb = takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb
let ?drop-sb = dropWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb

show ?thesis
proof (cases a ∈ dom $\mathcal{S}$)
case True
from shared-share-acquired [OF consis-sb True]
have a ∈ dom (share sb $\mathcal{S}$) ∪ acquired True sb $\mathcal{O}$**.**
thus ?thesis **by** auto
next
case False
from share-all-until-volatile-write-share-acquired [OF dist consis False a-in]
obtain j **where** j-bound: j < length ts **and**
rel: let (-,-,-,sb_{j},-,-,-) = ts!j
in a ∈ all-shared (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{j})

```
      by auto
    show ?thesis
    proof (cases j=i)
      case False
      with j-bound rel
      show ?thesis
        by blast
    next
      case True
      with rel ts-i have a ∈ all-shared ?take-sb
        by (auto simp add: Let-def)
      hence a ∈ all-shared sb
      using all-shared-append [of ?take-sb ?drop-sb]
        by auto
      from all-shared-share-acquired [OF consis-sb this]
      have a ∈ dom (share sb 𝒮) ∪ acquired True sb 𝒪.
      thus ?thesis
        by auto
    qed
  qed
qed
```

```
lemma acquired-all-shared-in:
  ⋀A. a ∈ acquired True sb A ⟹ a ∈ acquired True sb {} ∨ (a ∈ A ∧ a ∉ all-shared sb)
proof (induct sb)
    case Nil thus ?case by simp
next
  case (Cons x sb)
  show ?case
  proof (cases x)
    case (Write_sb volatile a' sop v A' L R)
    show ?thesis
    proof (cases volatile)
      case True
      note volatile=this
      from Cons.prems
      have a-in: a ∈ acquired True sb (A ∪ A' − R)
 by (clarsimp simp add: Write_sb True)
      show ?thesis
      proof (cases a ∈ A)
 case True
 from Cons.hyps [OF a-in]
 have a ∈ acquired True sb {} ∨ a ∈ A ∪ A' − R ∧ a ∉ all-shared sb.
 then show ?thesis
 proof
```

assume $a \in$ acquired True sb {}
from acquired-mono-in [OF this]
have $a \in$ acquired True sb $(A' - R)$ **by** auto
then show ?thesis
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ volatile True)
next
assume $a \in A \cup A' - R \wedge a \notin$ all-shared sb
then obtain $a \notin R$ $a \notin$ all-shared sb
by blast
then show ?thesis **by** (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ volatile True)
qed
next
case False
have $(A \cup A' - R) \subseteq A \cup (A' - R)$
by blast
from acquired-mono [OF this] a-in
have $a \in$ acquired True sb $(A \cup (A' - R))$ **by** blast
from acquired-union-notin-first [OF this False]
have $a \in$ acquired True sb $(A' - R)$**.**
then show ?thesis
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ True)
qed
next
case False
with Cons **show** ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ False)
qed
next
case $\text{Read}_{\mathsf{sb}}$
with Cons **show** ?thesis
by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
next
case $\text{Prog}_{\mathsf{sb}}$
with Cons **show** ?thesis
by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
next
case ($\text{Ghost}_{\mathsf{sb}}$ A' L R W)
from Cons.prems
have a-in: $a \in$ acquired True sb $(A \cup A' - R)$
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
show ?thesis
proof (cases $a \in A$)
case True
from Cons.hyps [OF a-in]
have $a \in$ acquired True sb $\{\} \vee a \in A \cup A' - R \wedge a \notin$ all-shared sb**.**
then show ?thesis
proof
assume $a \in$ acquired True sb {}
from acquired-mono-in [OF this]
have $a \in$ acquired True sb $(A' - R)$ **by** auto

then show ?thesis
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$ True)
next
assume a $\in$ A $\cup$ A$'$ $-$ R $\wedge$ a $\notin$ all-shared sb
then obtain a $\notin$ R a $\notin$ all-shared sb
by blast
then show ?thesis **by** (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$ True)
qed
next
case False
have (A $\cup$ A$'$ $-$ R) $\subseteq$ A $\cup$ (A$'$ $-$ R)
by blast
from acquired-mono [OF this] a-in
have a $\in$ acquired True sb (A $\cup$ (A$'$ $-$ R)) **by** blast
from acquired-union-notin-first [OF this False]
have a $\in$ acquired True sb (A$'$ $-$ R)**.**
then show ?thesis
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
qed
qed
qed

lemma all-shared-acquired-in: $\bigwedge$A. a $\in$ A $\Longrightarrow$ a $\notin$ all-shared sb $\Longrightarrow$ a $\in$ acquired True sb A
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a$'$ sop v A$'$ L R W)
show ?thesis
proof (cases volatile)
case True
show ?thesis
proof $-$
from Cons.prems **obtain** a-A: a $\in$ A **and** a-R: a $\notin$ R **and** a-sb: a $\notin$ all-shared sb
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ True)
from a-A a-R **have** a $\in$ A $\cup$ A$'$ $-$ R
by blast
from Cons.hyps [OF this a-sb]
show ?thesis
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ True)
qed
next
case False
with Cons **show** ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ False)
qed

next
case $\text{Read}_{\mathsf{sb}}$
with Cons **show** ?thesis
by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
next
case $\text{Prog}_{\mathsf{sb}}$
with Cons **show** ?thesis
by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
next
case $\text{Ghost}_{\mathsf{sb}}$
with Cons **show** ?thesis
by (auto simp add: $\text{Ghost}_{\mathsf{sb}}$)
qed
qed

lemma owned-share-acquired: $\bigwedge\mathcal{S}\ \mathcal{O}$. sharing-consistent $\mathcal{S}\ \mathcal{O}$ sb $\Longrightarrow$
a $\in \mathcal{O} \Longrightarrow$ a $\in$ dom (share sb $\mathcal{S}$) $\cup$ acquired True sb $\mathcal{O}$
proof (induct sb)
case Nil **thus** ?case **by** auto
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a′ sop v A L R W)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems **obtain**
a-owned: a $\in \mathcal{O}$ **and** A-shared-owns: A $\subseteq$ dom $\mathcal{S} \cup \mathcal{O}$ **and**
L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq \mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ True)
show ?thesis
proof (cases a $\in$ R)
case False **with** a-owned
have a $\in \mathcal{O} \cup$ A $-$ R
by auto
from Cons.hyps [OF consis′ this]
show ?thesis
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ volatile)
next
case True
from True L-A A-R **have** a $\in$ dom ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by auto
from shared-share-acquired [OF consis′ this]
show ?thesis
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ volatile True)
qed
next

case False
with Cons
show ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$)
qed
next
case $\text{Read}_{\mathsf{sb}}$
with Cons **show** ?thesis
by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
next
case $\text{Prog}_{\mathsf{sb}}$
with Cons **show** ?thesis
by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.prems **obtain**
a-owned: a $\in \mathcal{O}$ **and** A-shared-owns: A $\subseteq$ dom $\mathcal{S} \cup \mathcal{O}$ **and**
L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq \mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A − R) sb
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
show ?thesis
proof (cases a $\in$ R)
case False **with** a-owned
have a $\in \mathcal{O} \cup$ A − R
by auto
from Cons.hyps [OF consis′ this]
show ?thesis
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
next
case True
from True L-A A-R **have** a $\in$ dom ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by auto
from shared-share-acquired [OF consis′ this]
show ?thesis
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$ True)
qed
qed
qed

lemma outstanding-refs-non-volatile-$\text{Read}_{\mathsf{sb}}$-all-acquired:
$\bigwedge$m $\mathcal{S}$ $\mathcal{O}$ pending-write.
⟦reads-consistent pending-write $\mathcal{O}$ m sb;non-volatile-owned-or-read-only pending-write $\mathcal{S}$ $\mathcal{O}$ sb;
a $\in$ outstanding-refs is-non-volatile-$\text{Read}_{\mathsf{sb}}$ sb⟧
$\Longrightarrow$ a $\in$ $\mathcal{O}$ $\vee$ a $\in$ all-acquired sb $\vee$
a $\in$ read-only-reads $\mathcal{O}$ sb
proof (induct sb)
case Nil **thus** ?case **by** simp
next

case (Cons x sb)
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a′ sop v A L R W)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems **obtain**
non-vo: non-volatile-owned-or-read-only True ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
($\mathcal{O} \cup$ A − R) sb **and**
out-vol: outstanding-refs is-volatile-$\text{Read}_{\mathsf{sb}}$ sb = {} **and**
out: a ∈ outstanding-refs is-non-volatile-$\text{Read}_{\mathsf{sb}}$ sb
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ True)
show ?thesis
proof (cases a ∈ $\mathcal{O}$)
case True
show ?thesis
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ True volatile)
next
case False
from outstanding-non-volatile-$\text{Read}_{\mathsf{sb}}$-acquired-or-read-only-reads [OF non-vo out]
have a-in: a ∈ acquired-reads True sb ($\mathcal{O} \cup$ A − R) ∨
a ∈ read-only-reads ($\mathcal{O} \cup$ A − R) sb
by auto
with acquired-reads-all-acquired [of True sb ($\mathcal{O} \cup$ A − R)]
show ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ volatile)
qed
next
case False
with Cons **show** ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ False)
qed
next
case $\text{Read}_{\mathsf{sb}}$
with Cons **show** ?thesis
apply (clarsimp simp del: o-apply simp add: $\text{Read}_{\mathsf{sb}}$
acquired-takeWhile-non-volatile-$\text{Write}_{\mathsf{sb}}$ split: if-split-asm)
apply auto
done
next
case $\text{Prog}_{\mathsf{sb}}$
with Cons **show** ?thesis
by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L)
with Cons **show** ?thesis
by (auto simp add: $\text{Ghost}_{\mathsf{sb}}$)
qed

qed

lemma outstanding-refs-non-volatile-Read$_{sb}$-all-acquired-dropWhile:
assumes consis: reads-consistent pending-write $\mathcal{O}$ m sb
assumes nvo: non-volatile-owned-or-read-only pending-write $\mathcal{S}$ $\mathcal{O}$ sb
assumes out: a $\in$ outstanding-refs is-non-volatile-Read$_{sb}$ (dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb)
shows a $\in$ $\mathcal{O}$ $\vee$ a $\in$ all-acquired sb $\vee$
a $\in$ read-only-reads $\mathcal{O}$ sb
using outstanding-refs-append [of is-non-volatile-Read$_{sb}$ takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb
dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb]
outstanding-refs-non-volatile-Read$_{sb}$-all-acquired [OF consis nvo, of a] out
by (auto)

lemma share-commute:
$\bigwedge$L R $\mathcal{S}$ $\mathcal{O}$. $[\![$sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb;
all-shared sb $\cap$ L = {}; all-shared sb $\cap$ A = {}; all-acquired sb $\cap$ R = {};
all-unshared sb $\cap$ R = {}; all-shared sb $\cap$ R = {}$]\!]$ $\Longrightarrow$
(share sb ($\mathcal{S}$ $\oplus_{W}$ R $\ominus_{A}$ L)) =
(share sb $\mathcal{S}$) $\oplus_{W}$ R $\ominus_{A}$ L
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case (Write$_{sb}$ volatile a sop v A′ L′ R′ W′)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems **obtain**
L-prop: (R′ $\cup$ all-shared sb) $\cap$ L = {} **and**
A-prop: (R′ $\cup$ all-shared sb) $\cap$ A = {} **and**
R-acq-prop: (A′ $\cup$ all-acquired sb) $\cap$ R = {} **and**
R-prop:(L′ $\cup$ all-unshared sb) $\cap$ R = {} **and**
R-prop-sh: (R′ $\cup$ all-shared sb) $\cap$ R = {} **and**
A′-shared-owns: A′ $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ **and** L′-A′: L′ $\subseteq$ A′ **and** A′-R′: A′ $\cap$ R′ = {} **and**
R′-owns: R′ $\subseteq$ $\mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S}$ $\oplus_{W'}$ R′ $\ominus_{A'}$ L′) ($\mathcal{O}$ $\cup$ A′ − R′) sb
by (clarsimp simp add: Write$_{sb}$ volatile)

from L-prop **obtain** R′-L: R′ $\cap$ L = {} **and** acq-L: all-shared sb $\cap$ L = {}
by blast
from A-prop **obtain** R′-A: R′ $\cap$ A = {} **and** acq-A: all-shared sb $\cap$ A = {}

by blast

from R-acq-prop **obtain** A'-R: $A' \cap R = \{\}$ **and** acq-R:all-acquired sb $\cap$ R = {}

by blast

from R-prop **obtain** L'-R: $L' \cap R = \{\}$ **and** unsh-R: all-unshared sb $\cap$ R = {}

by blast

from R-prop-sh **obtain** R'-R: $R' \cap R = \{\}$ **and** sh-R: all-shared sb $\cap$ R = {}

by blast

from Cons.hyps [OF consis$'$ acq-L acq-A acq-R unsh-R sh-R]

have share sb $((\mathcal{S} \oplus_{W'} R' \ominus_{A'} L') \oplus_W R \ominus_A L)$ = share sb $(\mathcal{S} \oplus_{W'} R' \ominus_{A'} L') \oplus_W R \ominus_A L$.

moreover

from R'-L L'-R R'-R R'-A A'-R

have $((\mathcal{S} \oplus_W R \ominus_A L) \oplus_{W'} R' \ominus_{A'} L') = ((\mathcal{S} \oplus_{W'} R' \ominus_{A'} L') \oplus_W R \ominus_A L)$

apply −

apply (rule ext)

apply (clarsimp simp add: augment-shared-def restrict-shared-def)

apply (auto split: if-split-asm option.splits)

done

ultimately

have share sb $((\mathcal{S} \oplus_W R \ominus_A L) \oplus_{W'} R' \ominus_{A'} L')$ = share sb $(\mathcal{S} \oplus_{W'} R' \ominus_{A'} L') \oplus_W R \ominus_A L$

by simp

then

show ?thesis

by (clarsimp simp add: Write_{sb} volatile)

next

case False **with** Cons **show** ?thesis

by (clarsimp simp add: Write_{sb} False)

qed

next

case Read_{sb} **with** Cons **show** ?thesis

by (clarsimp simp add: Read_{sb})

next

case Prog_{sb} **with** Cons **show** ?thesis

by (clarsimp simp add: Prog_{sb})

next

case (Ghost_{sb} A' L' R' W')

from Cons.prems **obtain**

L-prop: $(R' \cup$ all-shared sb$) \cap L = \{\}$ **and**

A-prop: $(R' \cup$ all-shared sb$) \cap A = \{\}$ **and**

R-acq-prop: $(A' \cup$ all-acquired sb$) \cap R = \{\}$ **and**

R-prop:$(L' \cup$ all-unshared sb$) \cap R = \{\}$ **and**

R-prop-sh: $(R' \cup$ all-shared sb$) \cap R = \{\}$ **and**

A'-shared-owns: $A' \subseteq \text{dom } \mathcal{S} \cup \mathcal{O}$ **and** L'-A': $L' \subseteq A'$ **and** A'-R': $A' \cap R' = \{\}$ **and**

R'-owns: $R' \subseteq \mathcal{O}$ **and**

consis$'$: sharing-consistent $(\mathcal{S} \oplus_{W'} R' \ominus_{A'} L')$ $(\mathcal{O} \cup A' - R')$ sb

by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)

from L-prop **obtain** R'-L: $R' \cap L = \{\}$ **and** acq-L: all-shared sb $\cap$ L = {}
by blast
from A-prop **obtain** R'-A: $R' \cap A = \{\}$ **and** acq-A: all-shared sb $\cap$ A = {}
by blast
from R-acq-prop **obtain** A'-R: $A' \cap R = \{\}$ **and** acq-R:all-acquired sb $\cap$ R = {}
by blast
from R-prop **obtain** L'-R: $L' \cap R = \{\}$ **and** unsh-R: all-unshared sb $\cap$ R = {}
by blast
from R-prop-sh **obtain** R'-R: $R' \cap R = \{\}$ **and** sh-R: all-shared sb $\cap$ R = {}
by blast

from Cons.hyps [OF consis$'$ acq-L acq-A acq-R unsh-R sh-R]
have share sb $((\mathcal{S} \oplus_{\mathsf{W}'} R' \ominus_{\mathsf{A}'} L') \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L)$ = share sb $(\mathcal{S} \oplus_{\mathsf{W}'} R' \ominus_{\mathsf{A}'} L') \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L$.

moreover

from R'-L L'-R R'-R R'-A A'-R
have $((\mathcal{S} \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L) \oplus_{\mathsf{W}'} R' \ominus_{\mathsf{A}'} L') = ((\mathcal{S} \oplus_{\mathsf{W}'} R' \ominus_{\mathsf{A}'} L') \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L)$
apply −
apply (rule ext)
apply (clarsimp simp add: augment-shared-def restrict-shared-def)
apply (auto split: if-split-asm option.splits)
done

ultimately
have share sb $((\mathcal{S} \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L) \oplus_{\mathsf{W}'} R' \ominus_{\mathsf{A}'} L')$ = share sb $(\mathcal{S} \oplus_{\mathsf{W}'} R' \ominus_{\mathsf{A}'} L') \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L$
by simp
then
show ?thesis
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
qed
qed

lemma share-all-until-volatile-write-commute:
$\bigwedge$ $\mathcal{S}$ R L. ⟦ownership-distinct ts; sharing-consis $\mathcal{S}$ ts;
$\forall$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts $\longrightarrow$ ts!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-shared (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) $\cap$ L = {};
$\forall$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts $\longrightarrow$ ts!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-shared (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) $\cap$ A = {};
$\forall$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts $\longrightarrow$ ts!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-acquired (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) $\cap$ R = {};
$\forall$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts $\longrightarrow$ ts!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-unshared (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) $\cap$ R = {};
$\forall$i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts $\longrightarrow$ ts!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$

all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {}⟧
$\Longrightarrow$
share-all-until-volatile-write ts $\mathcal{S}$ $\oplus_{W}$ R $\ominus_{A}$ L = share-all-until-volatile-write ts ($\mathcal{S}$ $\oplus_{W}$ R $\ominus_{A}$ L)
proof (induct ts)
  case Nil
  thus ?case **by** simp
next
  case (Cons t ts)
  obtain p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb **where**
    t: t=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
    by (cases t)
  have dist: ownership-distinct (t#ts) **by** fact
  then interpret ownership-distinct t#ts**.**
  have consis: sharing-consis $\mathcal{S}$ (t#ts) **by** fact
  then interpret sharing-consis $\mathcal{S}$ t#ts**.**

  have L-prop: ∀ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length (t#ts) ⟶ (t#ts)!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ L = {} **by** fact
  hence L-prop′: ∀ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length (ts) ⟶ (ts)!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ L = {}
    by force
  have A-prop: ∀ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length (t#ts) ⟶ (t#ts)!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ A = {} **by** fact
  hence A-prop′: ∀ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length (ts) ⟶ (ts)!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ A = {}
    by force
  have R-prop-acq: ∀ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length (t#ts) ⟶ (t#ts)!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {} **by** fact
  hence R-prop-acq′: ∀ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length (ts) ⟶ (ts)!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶

        all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {}
    by force

  have R-prop: ∀ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length (t#ts) ⟶ (t#ts)!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        all-unshared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {} **by** fact
  hence R-prop′: ∀ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length (ts) ⟶ (ts)!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        all-unshared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {}
    by force

  have R-prop-sh: ∀ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length (t#ts) ⟶ (t#ts)!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {} **by** fact
  hence R-prop-sh′: ∀ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length (ts) ⟶ (ts)!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {}
    by force

  from ownership-distinct-tl [OF dist]

have dist′: ownership-distinct ts.

from sharing-consis-tl [OF consis]
have consis′: sharing-consis $\mathcal{S}$ ts.
then
interpret consis′: sharing-consis $\mathcal{S}$ ts.

from L-prop [rule-format, of 0 p is j sb $\mathcal{D}$ $\mathcal{O}$] t
have sh-L: all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) $\cap$ L = {}
  by simp

from A-prop [rule-format, of 0 p is j sb $\mathcal{D}$ $\mathcal{O}$] t
have sh-A: all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) $\cap$ A = {}
  by simp

from R-prop-acq [rule-format, of 0 p is j sb $\mathcal{D}$ $\mathcal{O}$] t
have acq-R: all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) $\cap$ R = {}
  by simp

from R-prop [rule-format, of 0 p is j sb $\mathcal{D}$ $\mathcal{O}$] t
have unsh-R: all-unshared (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) $\cap$ R = {}
  by simp

from R-prop-sh [rule-format, of 0 p is j sb $\mathcal{D}$ $\mathcal{O}$] t
have sh-R: all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) $\cap$ R = {}
  by simp

from sharing-consis [of 0, simplified, OF t]
have sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb.
from sharing-consistent-takeWhile [OF this]
have consis-sb: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb).

from share-commute [OF consis-sb sh-L sh-A acq-R unsh-R sh-R]
have share-eq:
  (share (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) =
    (share (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) $\mathcal{S}$) $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L.

let ?$\mathcal{S}'$ = (share (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) $\mathcal{S}$)

from freshly-shared-owned [OF consis-sb]
have fresh-owned: dom ?$\mathcal{S}'$ $-$ dom $\mathcal{S}$ $\subseteq$ $\mathcal{O}$.
from unshared-all-unshared [OF consis-sb] unshared-acquired-or-owned [OF consis-sb]
have unshared-acq-owned: dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$
        $\subseteq$ all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) $\cup$ $\mathcal{O}$
  by simp

have sep:
  $\forall$i < length ts. let (-,-,-,sb′,-,-,-) = ts!i in

all-acquired sb′ ∩ dom $\mathcal{S}$ − dom ?$\mathcal{S}'$ = {} ∧
all-unshared sb′ ∩ dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ = {}
proof −
{
fix i p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j_i sb_i
assume i-bound: i < length ts
assume ts-i: ts ! i = (p_i,is_i,j_i,sb_i,$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$)
have all-acquired sb_i ∩ dom $\mathcal{S}$ − dom ?$\mathcal{S}'$ = {} ∧
all-unshared sb_i ∩ dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ = {}
proof −
from ownership-distinct [of 0 Suc i] ts-i t i-bound
have dist: ($\mathcal{O}$ ∪ all-acquired sb) ∩ ($\mathcal{O}_i$ ∪ all-acquired sb_i) = {}
by force

from dist unshared-acq-owned all-acquired-takeWhile [of (Not ∘ is-volatile-Write$_{sb}$) sb]
have all-acquired sb_i ∩ dom $\mathcal{S}$ − dom ?$\mathcal{S}'$ = {}
by blast

moreover

from sharing-consis [of Suc i] ts-i i-bound
have sharing-consistent $\mathcal{S}$ $\mathcal{O}_i$ sb_i
by force
from unshared-acquired-or-owned [OF this]
have all-unshared sb_i ⊆ all-acquired sb_i ∪ $\mathcal{O}_i$**.**
with dist fresh-owned
have all-unshared sb_i ∩ dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ = {}
by blast

ultimately show ?thesis **by** simp
qed
}
thus ?thesis
by (fastforce simp add: Let-def)
qed

from consis′.sharing-consis-preservation [OF sep]
have sharing-consis′: sharing-consis (share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{S}$) ts**.**

from Cons.hyps [OF dist′ sharing-consis′ L-prop′ A-prop′ R-prop-acq′ R-prop′ R-prop-sh′]
have share-all-until-volatile-write ts ?$\mathcal{S}'$ ⊕$_W$ R ⊖$_A$ L =
share-all-until-volatile-write ts (?$\mathcal{S}'$ ⊕$_W$ R ⊖$_A$ L)**.**

then
have share-all-until-volatile-write ts
?$\mathcal{S}'$ ⊕$_W$ R ⊖$_A$ L =
share-all-until-volatile-write ts

(share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L))
    by (simp add: share-eq)
  then
  show ?case
    by (simp add: t)
qed

lemma share-append-Ghost$_{sb}$:
  $\bigwedge\mathcal{S}$. outstanding-refs is-volatile-Write$_{sb}$ sb = {} $\Longrightarrow$ (share (sb @ [Ghost$_{sb}$ A L R W]) $\mathcal{S}$) = (share sb $\mathcal{S}$) $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L
apply (induct sb)
apply simp
subgoal for a sb $\mathcal{S}$
apply (case-tac a)
apply auto
done
done

lemma share-append-Ghost$_{sb}$':
  $\bigwedge\mathcal{S}$. outstanding-refs is-volatile-Write$_{sb}$ sb $\neq$ {} $\Longrightarrow$
    (share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) (sb @ [Ghost$_{sb}$ A L R W])) $\mathcal{S}$) =
     (share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{S}$)
apply (induct sb)
apply simp
subgoal for a sb $\mathcal{S}$
apply (case-tac a)
apply force+
done
done

lemma share-all-until-volatile-write-append-Ghost$_{sb}$:
assumes no-out-VWrite$_{sb}$: outstanding-refs is-volatile-Write$_{sb}$ sb = {}
shows $\bigwedge\mathcal{S}$ i. ⟦ownership-distinct ts; sharing-consis $\mathcal{S}$ ts;
  i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
  ∀ j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts ⟶ i≠j ⟶ ts!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ L = {};
  ∀ j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts ⟶ i≠j ⟶ ts!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ A = {};
    ∀ j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts ⟶ i≠j ⟶ ts!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {};
    ∀ j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts ⟶ i≠j ⟶ ts!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        all-unshared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {};
    ∀ j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts ⟶ i≠j ⟶ ts!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {}⟧
  $\Longrightarrow$
  share-all-until-volatile-write (ts[i := (p', is',j', sb @ [Ghost$_{sb}$ A L R W], $\mathcal{D}$', $\mathcal{O}$')]) $\mathcal{S}$
        = share-all-until-volatile-write ts $\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L
proof (induct ts)
  case Nil
  thus ?case **by** simp

next
case (Cons t ts)
obtain p_t is_t $\mathcal{O}_t$ $\mathcal{R}_t$ $\mathcal{D}_t$ acq_t j_t sb_t **where**
t: $t=(p_t,is_t,j_t,sb_t,\mathcal{D}_t,\mathcal{O}_t,\mathcal{R}_t)$
by (cases t)
have dist: ownership-distinct (t#ts) **by** fact
then interpret ownership-distinct t#ts**.**
have consis: sharing-consis $\mathcal{S}$ (t#ts) **by** fact
then interpret sharing-consis $\mathcal{S}$ t#ts**.**

have L-prop: $\forall$ j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length (t#ts) $\longrightarrow$ i$\neq$j $\longrightarrow$ (t#ts)!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\cap$ L = {} **by** fact

have A-prop: $\forall$ j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length (t#ts) $\longrightarrow$ i$\neq$j $\longrightarrow$ (t#ts)!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\cap$ A = {} **by** fact

have R-prop-acq: $\forall$ j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length (t#ts) $\longrightarrow$ i$\neq$j$\longrightarrow$ (t#ts)!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\cap$ R = {} **by** fact
have R-prop: $\forall$ j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length (t#ts) $\longrightarrow$ i$\neq$j$\longrightarrow$ (t#ts)!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-unshared (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\cap$ R = {} **by** fact

have R-prop-sh: $\forall$ j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length (t#ts) $\longrightarrow$ i$\neq$j $\longrightarrow$ (t#ts)!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\cap$ R = {} **by** fact

from ownership-distinct-tl [OF dist]
have dist$'$: ownership-distinct ts**.**

from sharing-consis-tl [OF consis]
have consis$'$: sharing-consis $\mathcal{S}$ ts**.**
then
interpret consis$'$: sharing-consis $\mathcal{S}$ ts**.**

from sharing-consis [of 0, simplified, OF t]
have sharing-consistent $\mathcal{S}$ $\mathcal{O}_t$ sb_t **.**

from sharing-consistent-takeWhile [OF this]
have consis-sb: sharing-consistent $\mathcal{S}$ $\mathcal{O}_t$ (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_t)**.**

let ?$\mathcal{S}'$ = (share (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_t) $\mathcal{S}$)

from freshly-shared-owned [OF consis-sb]
have fresh-owned: dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ $\subseteq$ $\mathcal{O}_t$**.**
from unshared-all-unshared [OF consis-sb] unshared-acquired-or-owned [OF consis-sb]
have unshared-acq-owned: dom $\mathcal{S}$ − dom ?$\mathcal{S}'$

$\subseteq$ all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_t$) $\cup$ $\mathcal{O}_t$
by simp

have sep:
$\forall$i < length ts. let (-,-,-,sb$'$,-,-,-) = ts!i in
all-acquired sb$'$ $\cap$ dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$ = {} $\wedge$
all-unshared sb$'$ $\cap$ dom ?$\mathcal{S}'$ $-$ dom $\mathcal{S}$ = {}
proof $-$
{
fix i p$_i$ is$_i$ $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ acq$_i$ j$_i$ sb$_i$
assume i-bound: i < length ts
assume ts-i: ts ! i = (p$_i$,is$_i$,j$_i$,sb$_i$,$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$)
have all-acquired sb$_i$ $\cap$ dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$ = {} $\wedge$
all-unshared sb$_i$ $\cap$ dom ?$\mathcal{S}'$ $-$ dom $\mathcal{S}$ = {}
proof $-$
from ownership-distinct [of 0 Suc i] ts-i t i-bound
have dist: ($\mathcal{O}_t$ $\cup$ all-acquired sb$_t$) $\cap$ ($\mathcal{O}_i$ $\cup$ all-acquired sb$_i$) = {}
by force

from dist unshared-acq-owned all-acquired-takeWhile [of (Not $\circ$ is-volatile-Write$_{sb}$) sb$_t$]
have all-acquired sb$_i$ $\cap$ dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$ = {}
by blast

moreover

from sharing-consis [of Suc i] ts-i i-bound
have sharing-consistent $\mathcal{S}$ $\mathcal{O}_i$ sb$_i$
by force
from unshared-acquired-or-owned [OF this]
have all-unshared sb$_i$ $\subseteq$ all-acquired sb$_i$ $\cup$ $\mathcal{O}_i$.
with dist fresh-owned
have all-unshared sb$_i$ $\cap$ dom ?$\mathcal{S}'$ $-$ dom $\mathcal{S}$ = {}
by blast

ultimately show ?thesis **by** simp
qed
}
thus ?thesis
by (fastforce simp add: Let-def)
qed

from consis$'$.sharing-consis-preservation [OF sep]
have sharing-consis$'$: sharing-consis (share (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_t$) $\mathcal{S}$) ts.

show ?case
proof (cases i)
case 0

with t Cons.prems **have** eqs: p_t=p is_t=is $\mathcal{O}_t$=$\mathcal{O}$ $\mathcal{R}_t$=$\mathcal{R}$ j_t=j sb_t=sb $\mathcal{D}_t$=$\mathcal{D}$
  by auto

from no-out-$VWrite_{sb}$
have flush-all: takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb = sb
  by (auto simp add: outstanding-refs-conv)

from no-out-$VWrite_{sb}$
  have flush-all′: takeWhile (Not ∘ is-volatile-$Write_{sb}$) (sb@[$Ghost_{sb}$ A L R W]) = sb@[$Ghost_{sb}$ A L R W]
  by (auto simp add: outstanding-refs-conv)

have share-eq:
  (share (takeWhile (Not ∘ is-volatile-$Write_{sb}$) (sb @ [$Ghost_{sb}$ A L R W])) $\mathcal{S}$) =
    (share (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb) $\mathcal{S}$) $\oplus_W$ R $\ominus_A$ L
  apply (simp only: flush-all flush-all′)
  apply (rule share-append-$Ghost_{sb}$ [OF no-out-$VWrite_{sb}$])
  done

from L-prop 0 **have** L-prop′:
∀i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
  i < length ts ⟶
  ts ! i = (p, is,j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) ⟶
  all-shared (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb) ∩ L = {}
  apply clarsimp
  subgoal for i1 p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
  apply (drule-tac x=Suc i1 **in** spec)
  apply auto
  done
  done
from A-prop 0 **have** A-prop′:
∀i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
  i < length ts ⟶
  ts ! i = (p, is,j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) ⟶
  all-shared (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb) ∩ A = {}
  apply clarsimp
  subgoal for i1 p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
  apply (drule-tac x=Suc i1 **in** spec)
  apply auto
  done
  done
from R-prop-acq 0 **have** R-prop-acq′:
  ∀i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts ⟶ ts!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
    all-acquired (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb) ∩ R = {}
  apply clarsimp
  subgoal for i1 p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
  apply (drule-tac x=Suc i1 **in** spec)
  apply auto
  done

done
from R-prop 0
have R-prop′:
∀i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts ⟶ ts!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
all-unshared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {}
apply clarsimp
subgoal for i1 p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
apply (drule-tac x=Suc i1 in spec)
apply auto
done
done
from R-prop-sh 0 have R-prop-sh′:
∀i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts ⟶ ts!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {}
apply clarsimp
subgoal for i1 p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
apply (drule-tac x=Suc i1 in spec)
apply auto
done
done

from share-all-until-volatile-write-commute [OF dist′ sharing-consis′ L-prop′ A-prop′ R-prop-acq′ R-prop′
R-prop-sh′]

have share-all-until-volatile-write ts (share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{S}$ ⊕$_{W}$ R ⊖$_{A}$ L) =
share-all-until-volatile-write ts (share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_{t}$) $\mathcal{S}$) ⊕$_{W}$ R ⊖$_{A}$ L
by (simp add: eqs)
with share-eq
show ?thesis
by (clarsimp simp add: 0 t)
next
case (Suc k)
from L-prop Suc
have L-prop′: ∀j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length (ts) ⟶ k≠j ⟶ (ts)!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ L = {} by force

from A-prop Suc
have A-prop′: ∀j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length (ts) ⟶ k≠j ⟶ (ts)!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ A = {} by force
from R-prop-acq Suc have R-prop-acq′:
∀j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts ⟶ k≠j ⟶ ts!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {} by force

from R-prop Suc

have R-prop$'$:
$\forall$j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts $\longrightarrow$ k$\neq$j $\longrightarrow$ ts!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-unshared (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\cap$ R = {} **by** force

from R-prop-sh Suc **have** R-prop-sh$'$:
$\forall$j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts $\longrightarrow$ k$\neq$j $\longrightarrow$ ts!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\cap$ R = {} **by** force

from Cons.prems Suc **obtain** k-bound: k < length ts **and** ts-k: ts!k = (p, is,j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)
by auto

from Cons.hyps [OF dist$'$ sharing-consis$'$ k-bound ts-k L-prop$'$ A-prop$'$ R-prop-acq$'$ R-prop$'$ R-prop-sh$'$]
show ?thesis
by (clarsimp simp add: t Suc)
qed
qed

lemma share-domain-changes:
$\bigwedge\mathcal{S}$ $\mathcal{S}'$. a $\in$ all-shared sb $\cup$ all-unshared sb $\Longrightarrow$ share sb $\mathcal{S}'$ a = share sb $\mathcal{S}$ a
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case (Write$_{sb}$ volatile a$'$ sop v A L R W)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems **obtain** a-in: a $\in$ R $\cup$ all-shared sb $\cup$ L $\cup$ all-unshared sb
by (clarsimp simp add: Write$_{sb}$ True)
show ?thesis
proof (cases a $\in$ R)
case True
from True **have** ($\mathcal{S}'$ $\oplus_{W}$ R $\ominus_{A}$ L) a = ($\mathcal{S}$ $\oplus_{W}$ R $\ominus_{A}$ L) a
by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
from share-shared-eq [**where** $\mathcal{S}'$=$\mathcal{S}'$ $\oplus_{W}$ R $\ominus_{A}$ L **and** $\mathcal{S}$=$\mathcal{S}$ $\oplus_{W}$ R $\ominus_{A}$ L, OF this]
have share sb ($\mathcal{S}'$ $\oplus_{W}$ R $\ominus_{A}$ L) a = share sb ($\mathcal{S}$ $\oplus_{W}$ R $\ominus_{A}$ L) a
by auto
then show ?thesis
by (clarsimp simp add: Write$_{sb}$ volatile)
next

case False
note not-R = this
show ?thesis
proof (cases $a \in L$)
case True
from not-R True **have** $(\mathcal{S}' \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L)$ a = $(\mathcal{S} \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L)$ a
by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
from share-shared-eq [**where** $\mathcal{S}'=\mathcal{S}' \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L$ **and** $\mathcal{S}=\mathcal{S} \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L$, OF this]
have share sb $(\mathcal{S}' \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L)$ a = share sb $(\mathcal{S} \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L)$ a
by auto
then show ?thesis
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ volatile)
next
case False
with not-R a-in **have** $a \in$ all-shared sb $\cup$ all-unshared sb
by auto
from Cons.hyps [OF this]
show ?thesis **by** (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ volatile)
qed
qed
next
case False **with** Cons **show** ?thesis **by** (auto simp add: $\text{Write}_{\mathsf{sb}}$)
qed
next
case $\text{Read}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** (auto)
next
case $\text{Prog}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** (auto)
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.prems **obtain** a-in: $a \in R \cup$ all-shared sb $\cup$ L $\cup$ all-unshared sb
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
show ?thesis
proof (cases $a \in R$)
case True
from True **have** $(\mathcal{S}' \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L)$ a = $(\mathcal{S} \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L)$ a
by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
from share-shared-eq [**where** $\mathcal{S}'=\mathcal{S}' \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L$ **and** $\mathcal{S}=\mathcal{S} \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L$, OF this]
have share sb $(\mathcal{S}' \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L)$ a = share sb $(\mathcal{S} \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L)$ a
by auto
then show ?thesis
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
next
case False
note not-R = this
show ?thesis
proof (cases $a \in L$)
case True
from not-R True **have** $(\mathcal{S}' \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L)$ a = $(\mathcal{S} \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L)$ a
by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)

from share-shared-eq [**where** $\mathcal{S}'=\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L **and** $\mathcal{S}=\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L, OF this]
have share sb ($\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = share sb ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a
by auto
then show ?thesis
by (clarsimp simp add: $\mathrm{Ghost}_{\mathsf{sb}}$)
next
case False
with not-R a-in **have** a ∈ all-shared sb ∪ all-unshared sb
by auto
from Cons.hyps [OF this]
show ?thesis **by** (clarsimp simp add: $\mathrm{Ghost}_{\mathsf{sb}}$)
qed
qed
qed
qed

lemma share-domain-changesX:
$\bigwedge\mathcal{S}\ \mathcal{S}'$ X. ∀a ∈ X. $\mathcal{S}'$ a = $\mathcal{S}$ a
⟹ a ∈ all-shared sb ∪ all-unshared sb ∪ X ⟹ share sb $\mathcal{S}'$ a = share sb $\mathcal{S}$ a
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
then have shared-eq: ∀a ∈ X. $\mathcal{S}'$ a = $\mathcal{S}$ a
by auto
show ?case
proof (cases x)
case ($\mathrm{Write}_{\mathsf{sb}}$ volatile a′ sop v A L R W)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems **obtain** a-in: a ∈ R ∪ all-shared sb ∪ L ∪ all-unshared sb ∪ X
by (clarsimp simp add: $\mathrm{Write}_{\mathsf{sb}}$ True)
show ?thesis
proof (cases a ∈ R)
case True
from True **have** ($\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a
by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
from share-shared-eq [**where** $\mathcal{S}'=\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L **and** $\mathcal{S}=\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L, OF this]
have share sb ($\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = share sb ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a
by auto
then show ?thesis
by (clarsimp simp add: $\mathrm{Write}_{\mathsf{sb}}$ volatile)
next
case False
note not-R = this
show ?thesis
proof (cases a ∈ L)
case True

from not-R True **have** $(\mathcal{S}' \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L})$ a = $(\mathcal{S} \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L})$ a
  by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
 from share-shared-eq [**where** $\mathcal{S}'$=$\mathcal{S}' \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L}$ **and** $\mathcal{S}$=$\mathcal{S} \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L}$, OF this]
have share sb $(\mathcal{S}' \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L})$ a = share sb $(\mathcal{S} \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L})$ a
  by auto
then show ?thesis
  by (clarsimp simp add: $\mathrm{Write}_{\mathsf{sb}}$ volatile)
next
case False
from shared-eq **have** shared-eq$'$: $\forall$ a $\in$ X. $(\mathcal{S}' \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L})$ a = $(\mathcal{S} \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L})$ a
  by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
from False not-R a-in **have** a $\in$ all-shared sb $\cup$ all-unshared sb $\cup$ X
  by auto
from Cons.hyps [OF shared-eq$'$ this]
show ?thesis **by** (clarsimp simp add: $\mathrm{Write}_{\mathsf{sb}}$ volatile)
qed
qed
next
case False **with** Cons **show** ?thesis **by** (auto simp add: $\mathrm{Write}_{\mathsf{sb}}$)
qed
next
case $\mathrm{Read}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** (auto)
next
case $\mathrm{Prog}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** (auto)
next
case ($\mathrm{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.prems **obtain** a-in: a $\in$ R $\cup$ all-shared sb $\cup$ L $\cup$ all-unshared sb $\cup$ X
  by (clarsimp simp add: $\mathrm{Ghost}_{\mathsf{sb}}$)
show ?thesis
proof (cases a $\in$ R)
case True
from True **have** $(\mathcal{S}' \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L})$ a = $(\mathcal{S} \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L})$ a
  by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
from share-shared-eq [**where** $\mathcal{S}'$=$\mathcal{S}' \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L}$ **and** $\mathcal{S}$=$\mathcal{S} \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L}$, OF this]
have share sb $(\mathcal{S}' \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L})$ a = share sb $(\mathcal{S} \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L})$ a
  by auto
then show ?thesis
 by (clarsimp simp add: $\mathrm{Ghost}_{\mathsf{sb}}$)
next
case False
note not-R = this
show ?thesis
proof (cases a $\in$ L)
case True
from not-R True **have** $(\mathcal{S}' \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L})$ a = $(\mathcal{S} \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L})$ a
  by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
from share-shared-eq [**where** $\mathcal{S}'$=$\mathcal{S}' \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L}$ **and** $\mathcal{S}$=$\mathcal{S} \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L}$, OF this]
have share sb $(\mathcal{S}' \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L})$ a = share sb $(\mathcal{S} \oplus_{\mathsf{W}} \mathrm{R} \ominus_{\mathsf{A}} \mathrm{L})$ a
  by auto

then show ?thesis
by (clarsimp simp add: $\mathsf{Ghost_{sb}}$)
next
case False
from shared-eq have shared-eq$'$: $\forall$ a $\in$ X. ($\mathcal{S}'$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a
by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
from False not-R a-in have a $\in$ all-shared sb $\cup$ all-unshared sb $\cup$ X
by auto
from Cons.hyps [OF shared-eq$'$ this]
show ?thesis by (clarsimp simp add: $\mathsf{Ghost_{sb}}$)
qed
qed
qed
qed

lemma share-unchanged:
$\bigwedge\mathcal{S}$. a $\notin$ all-shared sb $\cup$ all-unshared sb $\cup$ all-acquired sb $\Longrightarrow$ share sb $\mathcal{S}$ a = $\mathcal{S}$ a
proof (induct sb)
case Nil thus ?case by simp
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\mathsf{Write_{sb}}$ volatile a$'$ sop v A L R W)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems obtain a-R: a $\notin$ R and a-L: a $\notin$ L and a-A: a $\notin$ A
and a$'$: a $\notin$ all-shared sb $\cup$ all-unshared sb $\cup$ all-acquired sb
by (clarsimp simp add: $\mathsf{Write_{sb}}$ True)
from Cons.hyps [OF a$'$]
have share sb ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a .
moreover
from a-R a-L a-A have ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = $\mathcal{S}$ a
by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
ultimately
show ?thesis
by (clarsimp simp add: $\mathsf{Write_{sb}}$ True)
next
case False with Cons show ?thesis by (auto simp add: $\mathsf{Write_{sb}}$)
qed
next
case $\mathsf{Read_{sb}}$ with Cons show ?thesis by (auto)
next
case $\mathsf{Prog_{sb}}$ with Cons show ?thesis by (auto)
next
case ($\mathsf{Ghost_{sb}}$ A L R W)
from Cons.prems obtain a-R: a $\notin$ R and a-L: a $\notin$ L and a-A: a $\notin$ A
and a$'$: a $\notin$ all-shared sb $\cup$ all-unshared sb $\cup$ all-acquired sb

by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
from Cons.hyps [OF a']
have share sb ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a **.**
moreover
from a-R a-L a-A **have** ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = $\mathcal{S}$ a
by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
ultimately
show ?thesis
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
qed
qed

lemma share-augment-release-commute:
assumes dist: (R ∪ L ∪ A) ∩ (all-shared sb ∪ all-unshared sb ∪ all-acquired sb) = {}
shows (share sb $\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = share sb ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
proof −
from dist **have** shared-eq: ∀ a ∈ all-acquired sb. ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = $\mathcal{S}$ a
by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
{
fix a
assume a-in: a ∈ all-shared sb ∪ all-unshared sb ∪ all-acquired sb
from share-domain-changesX [OF shared-eq this]
have share sb ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = share sb $\mathcal{S}$ a**.**
also
from dist a-in **have** ... = (share sb $\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a
by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
finally have share sb ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = (share sb $\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a**.**
}
moreover
{
fix a
assume a-notin: a ∉ all-shared sb ∪ all-unshared sb ∪ all-acquired sb
from share-unchanged [OF a-notin]
have share sb ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a**.**
moreover
from share-unchanged [OF a-notin]
have share sb $\mathcal{S}$ a = $\mathcal{S}$ a**.**
hence (share sb $\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a
by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
ultimately have share sb ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a = (share sb $\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) a
by simp
}
ultimately show ?thesis
apply −
apply (rule ext)
subgoal for x
apply (case-tac x ∈ all-shared sb ∪ all-unshared sb ∪ all-acquired sb)
apply auto
done

```
    done
  qed

lemma share-append-commute:
  ⋀ys 𝒮. (all-shared xs ∪ all-unshared xs ∪ all-acquired xs) ∩
          (all-shared ys ∪ all-unshared ys ∪ all-acquired ys) = {}
  ⟹ share xs (share ys 𝒮) = share ys (share xs 𝒮)
proof (induct xs)
  case Nil thus ?case by simp
next
  case (Cons x xs)
  show ?case
  proof (cases x)
    case (Write_sb volatile a sop v A L R W)
    show ?thesis
    proof (cases volatile)
      case True
      note volatile=this
      from Cons.prems have
        dist': (all-shared xs ∪ all-unshared xs ∪ all-acquired xs) ∩
               (all-shared ys ∪  all-unshared ys ∪ all-acquired ys) = {}
        apply (clarsimp  simp add: Write_sb True)
        apply blast
        done
      from Cons.prems have
        dist: (R ∪ L ∪ A) ∩ (all-shared ys ∪ all-unshared ys ∪ all-acquired ys) = {}
        apply (clarsimp  simp add: Write_sb True)
        apply blast
        done
      from share-augment-release-commute [OF dist]
      have (share ys 𝒮 ⊕_W R ⊖_A L) = share ys (𝒮 ⊕_W R ⊖_A L).

      with Cons.hyps [OF dist']
      show ?thesis
        by (clarsimp simp add: Write_sb True)
    next
      case False with Cons show ?thesis
        by (clarsimp simp add: Write_sb False)
    qed
  next
    case Read_sb with Cons show ?thesis by auto
  next
    case Prog_sb with Cons show ?thesis by auto
  next
    case (Ghost_sb A L R W)
    from Cons.prems have
      dist': (all-shared xs ∪ all-unshared xs ∪ all-acquired xs) ∩
             (all-shared ys ∪  all-unshared ys ∪ all-acquired ys) = {}
      apply (clarsimp  simp add: Ghost_sb)
      apply blast
```

done
from Cons.prems **have**
dist: (R ∪ L ∪ A) ∩ (all-shared ys ∪ all-unshared ys ∪ all-acquired ys) = {}
apply (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
apply blast
done
from share-augment-release-commute [OF dist]
have (share ys $\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = share ys ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)**.**

with Cons.hyps [OF dist′]
show ?thesis
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
qed
qed

lemma share-append-commute′:
assumes dist: (all-shared xs ∪ all-unshared xs ∪ all-acquired xs) ∩
(all-shared ys ∪ all-unshared ys ∪ all-acquired ys) = {}
shows share (ys@xs) $\mathcal{S}$ = share (xs@ys) $\mathcal{S}$
proof −
from share-append-commute [OF dist] share-append [of xs ys] share-append [of ys xs]
show ?thesis
by simp
qed

lemma share-all-until-volatile-write-share-commute:
shows $\bigwedge$ $\mathcal{S}$ (sb′::′a memref list). ⟦ownership-distinct ts; sharing-consis $\mathcal{S}$ ts;
∀i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j (sb::′a memref list). i < length ts
⟶ ts!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
(all-shared (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) ∪
all-unshared (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) ∪
all-acquired (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb)) ∩
(all-shared sb′ ∪ all-unshared sb′ ∪ all-acquired sb′) = {}⟧
⟹
share-all-until-volatile-write ts (share sb′ $\mathcal{S}$) =
share sb′ (share-all-until-volatile-write ts $\mathcal{S}$)
proof (induct ts)
case Nil
thus ?case **by** simp
next
case (Cons t ts)
obtain p_{t} is_{t} $\mathcal{O}_{\mathsf{t}}$ $\mathcal{R}_{\mathsf{t}}$ $\mathcal{D}_{\mathsf{t}}$ j_{t} sb_{t} **where**
t: t=(p_{t},is_{t},j_{t},sb_{t},$\mathcal{D}_{\mathsf{t}}$,$\mathcal{O}_{\mathsf{t}}$,$\mathcal{R}_{\mathsf{t}}$)
by (cases t)

let ?take = (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{t})
have dist: ownership-distinct (t#ts) **by** fact
then interpret ownership-distinct t#ts **.**
have consis: sharing-consis $\mathcal{S}$ (t#ts) **by** fact
then interpret sharing-consis $\mathcal{S}$ t#ts **.**

have dist-prop: $\forall$ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length (t#ts)
$\longrightarrow$ (t#ts)!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
(all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) $\cup$
all-unshared (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) $\cup$
all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb)) $\cap$
(all-shared sb$'$ $\cup$ all-unshared sb$'$ $\cup$ all-acquired sb$'$) = {} **by** fact
from dist-prop [rule-format, of 0] t
have dist-t: (all-shared ?take $\cup$ all-unshared ?take $\cup$ all-acquired ?take) $\cap$
(all-shared sb$'$ $\cup$ all-unshared sb$'$ $\cup$ all-acquired sb$'$) = {}
apply clarsimp
done
from dist-prop **have**
dist-prop$'$:$\forall$ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts
$\longrightarrow$ ts!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
(all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) $\cup$
all-unshared (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb) $\cup$
all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb)) $\cap$
(all-shared sb$'$ $\cup$ all-unshared sb$'$ $\cup$ all-acquired sb$'$) = {}
apply (clarsimp)
subgoal for i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
apply (drule-tac x=Suc i **in** spec)
apply clarsimp
done
done

from ownership-distinct-tl [OF dist]
have dist$'$: ownership-distinct ts**.**

from sharing-consis-tl [OF consis]
have consis$'$: sharing-consis $\mathcal{S}$ ts**.**
then
interpret consis$'$: sharing-consis $\mathcal{S}$ ts **.**

from sharing-consis [of 0, simplified, OF t]
have sharing-consistent $\mathcal{S}$ $\mathcal{O}_{\mathsf{t}}$ sb$_{\mathsf{t}}$ **.**

from sharing-consistent-takeWhile [OF this]
have consis-sb: sharing-consistent $\mathcal{S}$ $\mathcal{O}_{\mathsf{t}}$?take**.**

let ?$\mathcal{S}'$ = (share ?take $\mathcal{S}$)

from freshly-shared-owned [OF consis-sb]
have fresh-owned: dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ $\subseteq$ $\mathcal{O}_{\mathsf{t}}$**.**
from unshared-all-unshared [OF consis-sb] unshared-acquired-or-owned [OF consis-sb]
have unshared-acq-owned: dom $\mathcal{S}$ − dom ?$\mathcal{S}'$
$\subseteq$ all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_{\mathsf{t}}$) $\cup$ $\mathcal{O}_{\mathsf{t}}$
by simp

have sep:
  $\forall$ i < length ts. let (-,-,-,sb$'$,-,-,-) = ts!i in
      all-acquired sb$'$ $\cap$ dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$ = {} $\wedge$
      all-unshared sb$'$ $\cap$ dom ?$\mathcal{S}'$ $-$ dom $\mathcal{S}$ = {}
proof $-$
  {
    fix i p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ acq_i j_i sb_i
    assume i-bound: i < length ts
    assume ts-i: ts ! i = (p_i,is_i,j_i,sb_i,$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$)
    have all-acquired sb_i $\cap$ dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$ = {} $\wedge$
        all-unshared sb_i $\cap$ dom ?$\mathcal{S}'$ $-$ dom $\mathcal{S}$ = {}
    proof $-$
from ownership-distinct [of 0 Suc i] ts-i t i-bound
have dist: ($\mathcal{O}_t$ $\cup$ all-acquired sb_t) $\cap$ ($\mathcal{O}_i$ $\cup$ all-acquired sb_i) = {}
  by force

from dist unshared-acq-owned all-acquired-takeWhile [of (Not $\circ$ is-volatile-$Write_{sb}$) sb_t]
have all-acquired sb_i $\cap$ dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$ = {}
  by blast

moreover

from sharing-consis [of Suc i] ts-i i-bound
have sharing-consistent $\mathcal{S}$ $\mathcal{O}_i$ sb_i
  by force
from unshared-acquired-or-owned [OF this]
have all-unshared sb_i $\subseteq$ all-acquired sb_i $\cup$ $\mathcal{O}_i$.
with dist fresh-owned
have all-unshared sb_i $\cap$ dom ?$\mathcal{S}'$ $-$ dom $\mathcal{S}$ = {}
  by blast

ultimately show ?thesis **by** simp
    qed
  }
  thus ?thesis
    by (fastforce simp add: Let-def)
qed

from consis$'$.sharing-consis-preservation [OF sep]
have sharing-consis$'$: sharing-consis ?$\mathcal{S}'$ ts.

have share-all-until-volatile-write ts (share ?take (share sb$'$ $\mathcal{S}$)) =
    share sb$'$ (share-all-until-volatile-write ts (share ?take $\mathcal{S}$))
proof $-$
  from share-append-commute [OF dist-t]
  have (share ?take (share sb$'$ $\mathcal{S}$)) = (share sb$'$ (share ?take $\mathcal{S}$)) .
  then
  have share-all-until-volatile-write ts (share ?take (share sb$'$ $\mathcal{S}$)) =
      share-all-until-volatile-write ts (share sb$'$ (share ?take $\mathcal{S}$))

    by (simp)
   also
   from Cons.hyps [OF dist′ sharing-consis′ dist-prop′]
   have ... = share sb′ (share-all-until-volatile-write ts (share ?take $\mathcal{S}$)).
   finally show ?thesis .
  qed
  then show ?case
   by (clarsimp simp add: t)
qed

lemma all-shared-takeWhile-subset: all-shared (takeWhile P sb) ⊆ all-shared sb
using all-shared-append [of (takeWhile P sb) (dropWhile P sb)]
  by auto
lemma all-shared-dropWhile-subset: all-shared (dropWhile P sb) ⊆ all-shared sb
using all-shared-append [of (takeWhile P sb) (dropWhile P sb)]
  by auto

lemma all-unshared-takeWhile-subset: all-unshared (takeWhile P sb) ⊆ all-unshared sb
using all-unshared-append [of (takeWhile P sb) (dropWhile P sb)]
  by auto
lemma all-unshared-dropWhile-subset: all-unshared (dropWhile P sb) ⊆ all-unshared sb
using all-unshared-append [of (takeWhile P sb) (dropWhile P sb)]
  by auto

lemma all-acquired-takeWhile-subset: all-acquired (takeWhile P sb) ⊆ all-acquired sb
using all-acquired-append [of (takeWhile P sb) (dropWhile P sb)]
  by auto
lemma all-acquired-dropWhile-subset: all-acquired (dropWhile P sb) ⊆ all-acquired sb
using all-acquired-append [of (takeWhile P sb) (dropWhile P sb)]
  by auto

lemma share-all-until-volatile-write-flush-commute:
assumes takeWhile-empty: (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb) = []
shows ⋀ $\mathcal{S}$ R L W A i. ⟦ownership-distinct ts; sharing-consis $\mathcal{S}$ ts; i < length ts;
    ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
    ∀i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j (sb::′a memref list). i < length ts
      ⟶ ts!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        (all-shared (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb) ∪
         all-unshared (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb) ∪
         all-acquired (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb)) ∩
        (all-shared (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb′) ∪
         all-unshared (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb′) ∪
         all-acquired (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb′)) = {};
    ∀j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j (sb::′a memref list). j < length ts ⟶ i≠j
      ⟶ ts!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        (all-shared sb ∪ all-unshared sb ∪ all-acquired sb) ∩
        (R ∪ L ∪ A) = {}⟧
 ⟹

share-all-until-volatile-write (ts[i :=(p′,is′,j′,sb′,$\mathcal{D}$′,$\mathcal{O}$′,$\mathcal{R}$′)]) ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) =
share (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb′) (share-all-until-volatile-write ts $\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
proof (induct ts)
  case Nil
  thus ?case **by** simp
next
  case (Cons t ts)
  obtain p_t is_t $\mathcal{O}_t$ $\mathcal{R}_t$ $\mathcal{D}_t$ j_t sb_t **where**
    t: t=(p_t,is_t,j_t,sb_t,$\mathcal{D}_t$,$\mathcal{O}_t$,$\mathcal{R}_t$)
    by (cases t)

  let ?take = (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb_t)
  let ?take-sb′ = (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb′)
  let ?drop = (dropWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb_t)
  have dist: ownership-distinct (t#ts) **by** fact
  then interpret ownership-distinct t#ts **.**
  have consis: sharing-consis $\mathcal{S}$ (t#ts) **by** fact
  then interpret sharing-consis $\mathcal{S}$ t#ts **.**
  have dist-prop: ∀i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length (t#ts)
      ⟶ (t#ts)!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        (all-shared (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb) ∪
         all-unshared (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb) ∪
         all-acquired (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb)) ∩
         (all-shared ?take-sb′ ∪ all-unshared ?take-sb′ ∪ all-acquired ?take-sb′) =
{} **by** fact
  from dist-prop [rule-format, of 0] t
  have dist-t: (all-shared ?take ∪ all-unshared ?take ∪ all-acquired ?take) ∩
      (all-shared ?take-sb′ ∪ all-unshared ?take-sb′ ∪ all-acquired ?take-sb′) = {}
    apply clarsimp
    done
  from dist-prop **have**
  dist-prop′:∀i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts
      ⟶ ts!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        (all-shared (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb) ∪
         all-unshared (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb) ∪
         all-acquired (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb)) ∩
        (all-shared ?take-sb′ ∪ all-unshared ?take-sb′ ∪ all-acquired ?take-sb′) = {}
    apply (clarsimp)
    subgoal for i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
    apply (drule-tac x=Suc i **in** spec)
    apply clarsimp
    done
    done
  have dist-prop-R-L-A: ∀j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length (t#ts) ⟶ i ≠ j
      ⟶ (t#ts)!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        (all-shared sb ∪ all-unshared sb ∪ all-acquired sb) ∩
        (R ∪ L ∪ A) = {} **by** fact

from ownership-distinct-tl [OF dist]
have dist′: ownership-distinct ts.

from sharing-consis-tl [OF consis]
have consis′: sharing-consis $\mathcal{S}$ ts.
then
interpret consis′: sharing-consis $\mathcal{S}$ ts .

from sharing-consis [of 0, simplified, OF t]
have sharing-consistent $\mathcal{S}$ $\mathcal{O}_t$ sb_t .

from sharing-consistent-takeWhile [OF this]
have consis-sb: sharing-consistent $\mathcal{S}$ $\mathcal{O}_t$ (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_t).

have aargh: (Not ∘ is-volatile-$Write_{sb}$) = (λa. ¬ is-volatile-$Write_{sb}$ a)
  by (rule ext) auto

show ?case
proof (cases i)
  case 0

  with t Cons.prems **have** eqs: p_t=p is_t=is $\mathcal{O}_t$=$\mathcal{O}$ $\mathcal{R}_t$=$\mathcal{R}$ j_t=j sb_t=sb $\mathcal{D}_t$=$\mathcal{D}$
    by auto

  let ?$\mathcal{S}'$ = $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L

  from dist-prop-R-L-A 0 **have**
    dist-prop-R-L-A′:∀ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts
      ⟶ ts!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
        (all-shared sb ∪ all-unshared sb ∪ all-acquired sb) ∩
        (R ∪ L ∪ A) = {}
    apply (clarsimp)
    subgoal for i1 p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
    apply (drule-tac x=Suc i1 **in** spec)
    apply clarsimp
    done
    done
  then
  have dist-prop-R-L-A″:∀ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts
    ⟶ ts!i=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
    (all-shared (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb) ∪ all-unshared (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb) ∪
    all-acquired (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb)) ∩
    (R ∪ L ∪ A) = {}
    apply (clarsimp)
    subgoal for i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
      apply (cut-tac sb=sb **in** all-shared-takeWhile-subset [**where** P=Not ∘ is-volatile-$Write_{sb}$])

apply (cut-tac sb=sb **in** all-unshared-takeWhile-subset [**where** P=Not ∘ is-volatile-Write$_{sb}$])
apply (cut-tac sb=sb **in** all-acquired-takeWhile-subset [**where** P=Not ∘ is-volatile-Write$_{sb}$])
apply fastforce
done
done

have sep: ∀i<length ts.
let (-, -, -, sb, -, -, -) = ts ! i
in ∀a∈all-acquired sb. ?$\mathcal{S}'$ a = $\mathcal{S}$ a
proof –
{
fix i p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ acq_i j_i sb_i a
assume i-bound: i < length ts
assume ts-i: ts ! i = (p_i,is_i,j_i,sb_i,$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$)
assume a-in: a ∈ all-acquired sb_i
have ?$\mathcal{S}'$ a = $\mathcal{S}$ a
proof –
from dist-prop-R-L-A′ [rule-format, OF i-bound ts-i] a-in
show ?thesis
by (auto simp add: augment-shared-def restrict-shared-def split: option.splits)
qed
}
thus ?thesis **by** auto
qed
from consis′.sharing-consis-shared-exchange [OF sep]
have sharing-consis′: sharing-consis ?$\mathcal{S}'$ ts**.**

from share-all-until-volatile-write-share-commute [of ts ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′), OF dist′ sharing-consis′ dist-prop′]

have share-all-until-volatile-write ts (share ?take-sb′ ?$\mathcal{S}'$) =
share ?take-sb′ (share-all-until-volatile-write ts ?$\mathcal{S}'$) **.**

moreover

from dist-prop-R-L-A″
have (share-all-until-volatile-write ts ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) =
(share-all-until-volatile-write ts $\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)

apply –
apply (rule share-all-until-volatile-write-commute [OF dist′ consis′, of L A R W,symmetric])
apply (clarsimp,blast)+
done
ultimately
show ?thesis
using takeWhile-empty
by (clarsimp simp add: t 0 aargh eqs)

next
case (Suc k)
from Cons.prems Suc **obtain** k-bound: k < length ts **and** ts-k: ts!k = (p, is,j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)
by auto

let ?$\mathcal{S}'$ = (share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_t$) $\mathcal{S}$)
from freshly-shared-owned [OF consis-sb]
have fresh-owned: dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ ⊆ $\mathcal{O}_t$.
from unshared-all-unshared [OF consis-sb] unshared-acquired-or-owned [OF consis-sb]
have unshared-acq-owned: dom $\mathcal{S}$ − dom ?$\mathcal{S}'$
⊆ all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_t$) ∪ $\mathcal{O}_t$
by simp

from freshly-shared-owned [OF consis-sb]
have fresh-owned: dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ ⊆ $\mathcal{O}_t$.
from unshared-all-unshared [OF consis-sb] unshared-acquired-or-owned [OF consis-sb]
have unshared-acq-owned: dom $\mathcal{S}$ − dom ?$\mathcal{S}'$
⊆ all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_t$) ∪ $\mathcal{O}_t$
by simp

have sep:
∀ i < length ts. let (-,-,-,sb′,-,-,-) = ts!i in
all-acquired sb′ ∩ dom $\mathcal{S}$ − dom ?$\mathcal{S}'$ = {} ∧
all-unshared sb′ ∩ dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ = {}
proof −
{
fix i p$_i$ is$_i$ $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ acq$_i$ j$_i$ sb$_i$
assume i-bound: i < length ts
assume ts-i: ts ! i = (p$_i$,is$_i$,j$_i$,sb$_i$,$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$)
have all-acquired sb$_i$ ∩ dom $\mathcal{S}$ − dom ?$\mathcal{S}'$ = {} ∧
all-unshared sb$_i$ ∩ dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ = {}
proof −
from ownership-distinct [of 0 Suc i] ts-i t i-bound
have dist: ($\mathcal{O}_t$ ∪ all-acquired sb$_t$) ∩ ($\mathcal{O}_i$ ∪ all-acquired sb$_i$) = {}
by force

from dist unshared-acq-owned all-acquired-takeWhile [of (Not ∘ is-volatile-Write$_{sb}$) sb$_t$]
have all-acquired sb$_i$ ∩ dom $\mathcal{S}$ − dom ?$\mathcal{S}'$ = {}
by blast

moreover

from sharing-consis [of Suc i] ts-i i-bound
have sharing-consistent $\mathcal{S}$ $\mathcal{O}_i$ sb$_i$
by force

from unshared-acquired-or-owned [OF this]
have all-unshared $sb_i \subseteq$ all-acquired $sb_i \cup \mathcal{O}_i$.
with dist fresh-owned
have all-unshared $sb_i \cap$ dom $?\mathcal{S}' -$ dom $\mathcal{S} = \{\}$
by blast

ultimately show ?thesis **by** simp
qed
}
thus ?thesis
by (fastforce simp add: Let-def)
qed
from consis$'$.sharing-consis-preservation [OF sep]
have sharing-consis$'$: sharing-consis $?\mathcal{S}'$ ts.

from dist-prop-R-L-A [rule-format, of 0] Suc t
have dist-t-R-L-A: (all-shared $sb_t \cup$ all-unshared $sb_t \cup$ all-acquired $sb_t) \cap$
$(R \cup L \cup A) = \{\}$
apply clarsimp
done
from dist-t-R-L-A
have $(R \cup L \cup A) \cap$ (all-shared ?take $\cup$ all-unshared ?take $\cup$ all-acquired ?take) $= \{\}$
using all-shared-append [of ?take ?drop] all-unshared-append [of ?take ?drop] all-acquired-append [of ?take ?drop]
by auto

from share-augment-release-commute [OF this]
have share ?take $\mathcal{S} \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L =$ share ?take $(\mathcal{S} \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L)$.
moreover

from dist-prop-R-L-A Suc
have $\forall$j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j $<$ length (ts) $\longrightarrow$ k $\neq$ j
$\longrightarrow$ (ts)!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
(all-shared sb $\cup$ all-unshared sb $\cup$ all-acquired sb) $\cap$
$(R \cup L \cup A) = \{\}$
apply (clarsimp)
subgoal for j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
apply (drule-tac x=Suc j **in** spec)
apply clarsimp
done
done
note Cons.hyps [OF dist$'$ sharing-consis$'$ k-bound ts-k dist-prop$'$ this, of W]
ultimately
show ?thesis
by (clarsimp simp add: t Suc)
qed
qed

lemma share-all-until-volatile-write-Ghost$_{sb}$-commute:
shows $\bigwedge$ $\mathcal{S}$ i. ⟦ownership-distinct ts; sharing-consis $\mathcal{S}$ ts; i < length ts;
ts!i = (p,is,j,Ghost$_{sb}$ A L R W#sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$);
∀j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts ⟶ i≠j ⟶ ts!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
(all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∪ all-unshared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∪ all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)) ∩

(R ∪ L ∪ A) = {}⟧
⟹
share-all-until-volatile-write (ts[i :=(p′,is′,j′,sb,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)]) ($\mathcal{S}$ $\oplus_{W}$ R $\ominus_{A}$ L) =
share-all-until-volatile-write ts $\mathcal{S}$
proof (induct ts)
case Nil
thus ?case **by** simp
next
case (Cons t ts)
obtain p_t is_t $\mathcal{O}_t$ $\mathcal{R}_t$ $\mathcal{D}_t$ j_t sb_t **where**
t: t=(p_t,is_t,j_t,sb_t,$\mathcal{D}_t$,$\mathcal{O}_t$,$\mathcal{R}_t$)
by (cases t)
have dist: ownership-distinct (t#ts) **by** fact
then interpret ownership-distinct t#ts **.**
have consis: sharing-consis $\mathcal{S}$ (t#ts) **by** fact
then interpret sharing-consis $\mathcal{S}$ t#ts **.**
have dist-prop: ∀j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length (t#ts) ⟶ i≠j ⟶ (t#ts)!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
(all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∪ all-unshared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∪ all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)) ∩

(R ∪ L ∪ A) = {} **by** fact

from ownership-distinct-tl [OF dist]
have dist′: ownership-distinct ts**.**

from sharing-consis-tl [OF consis]
have consis′: sharing-consis $\mathcal{S}$ ts**.**
then
interpret consis′: sharing-consis $\mathcal{S}$ ts **.**

from sharing-consis [of 0, simplified, OF t]
have sharing-consistent $\mathcal{S}$ $\mathcal{O}_t$ sb_t **.**

from sharing-consistent-takeWhile [OF this]
have consis-sb: sharing-consistent $\mathcal{S}$ $\mathcal{O}_t$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_t)**.**

let ?$\mathcal{S}'$ = (share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_t) $\mathcal{S}$)

from freshly-shared-owned [OF consis-sb]
have fresh-owned: dom ?$\mathcal{S}'$ − dom $\mathcal{S}$ ⊆ $\mathcal{O}_t$**.**
from unshared-all-unshared [OF consis-sb] unshared-acquired-or-owned [OF consis-sb]
have unshared-acq-owned: dom $\mathcal{S}$ − dom ?$\mathcal{S}'$

$\subseteq$ all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_t$) $\cup$ $\mathcal{O}_t$
by simp

have sep:
$\forall$i < length ts. let (-,-,-,sb',-,-,-) = ts!i in
all-acquired sb' $\cap$ dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$ = {} $\wedge$
all-unshared sb' $\cap$ dom ?$\mathcal{S}'$ $-$ dom $\mathcal{S}$ = {}
proof $-$
{
fix i p$_i$ is$_i$ $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j$_i$ sb$_i$
assume i-bound: i < length ts
assume ts-i: ts ! i = (p$_i$,is$_i$,j$_i$,sb$_i$,$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$)
have all-acquired sb$_i$ $\cap$ dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$ = {} $\wedge$
all-unshared sb$_i$ $\cap$ dom ?$\mathcal{S}'$ $-$ dom $\mathcal{S}$ = {}
proof $-$
from ownership-distinct [of 0 Suc i] ts-i t i-bound
have dist: ($\mathcal{O}_t$ $\cup$ all-acquired sb$_t$) $\cap$ ($\mathcal{O}_i$ $\cup$ all-acquired sb$_i$) = {}
by force

from dist unshared-acq-owned all-acquired-takeWhile [of (Not $\circ$ is-volatile-Write$_{sb}$) sb$_t$]
have all-acquired sb$_i$ $\cap$ dom $\mathcal{S}$ $-$ dom ?$\mathcal{S}'$ = {}
by blast

moreover

from sharing-consis [of Suc i] ts-i i-bound
have sharing-consistent $\mathcal{S}$ $\mathcal{O}_i$ sb$_i$
by force
from unshared-acquired-or-owned [OF this]
have all-unshared sb$_i$ $\subseteq$ all-acquired sb$_i$ $\cup$ $\mathcal{O}_i$.
with dist fresh-owned
have all-unshared sb$_i$ $\cap$ dom ?$\mathcal{S}'$ $-$ dom $\mathcal{S}$ = {}
by blast

ultimately show ?thesis **by** simp
qed
}
thus ?thesis
by (fastforce simp add: Let-def)
qed

from consis'.sharing-consis-preservation [OF sep]
have sharing-consis': sharing-consis (share (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_t$) $\mathcal{S}$) ts.

show ?case
proof (cases i)
case 0

with t Cons.prems **have** eqs: p_t=p is_t=is $\mathcal{O}_t$=$\mathcal{O}$ $\mathcal{R}_t$=$\mathcal{R}$ j_t=j sb_t=$Ghost_{sb}$ A L R W#sb $\mathcal{D}_t$=$\mathcal{D}$
by auto

show ?thesis
by (clarsimp simp add: 0 t eqs)
next
case (Suc k)
from Cons.prems Suc **obtain** k-bound: k < length ts **and** ts-k: ts!k = (p, is,j, $Ghost_{sb}$ A L R W#sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)
by auto

from dist-prop Suc
have dist-prop′: ∀ j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts ⟶ k≠j ⟶ ts!j=(p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟶
(all-shared (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb) ∪ all-unshared (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb) ∪ all-acquired (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb)) ∩

(R ∪ L ∪ A) = {}
apply (clarsimp)
subgoal for j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
apply (drule-tac x=Suc j **in** spec)
apply auto
done
done

from Cons.hyps [OF dist′ sharing-consis′ k-bound ts-k dist-prop′]
have share-all-until-volatile-write (ts[k := (p′, is′, j′, sb, $\mathcal{D}'$, $\mathcal{O}'$, $\mathcal{R}'$)])
(share (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_t) $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) =
share-all-until-volatile-write ts
(share (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_t) $\mathcal{S}$) **.**

moreover
from dist-prop [rule-format, of 0 p_t is_t j_t sb_t $\mathcal{D}_t$ $\mathcal{O}_t$ $\mathcal{R}_t$] t Suc
have (R ∪ L ∪ A) ∩ (all-shared (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_t) ∪ all-unshared (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_t) ∪ all-acquired (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_t)) = {}
apply clarsimp
apply blast
done
from share-augment-release-commute [OF this]
have share (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_t) $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L =
share (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_t) ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)**.**
ultimately
show ?thesis
by (clarsimp simp add: Suc t)
qed
qed

lemma share-all-until-volatile-write-update-sb:
assumes congr: $\bigwedge$S. share (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$'$) S = share (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) S
shows $\bigwedge\mathcal{S}$ i. $[\![$i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)$]\!]$
$\Longrightarrow$
share-all-until-volatile-write ts $\mathcal{S}$ =
share-all-until-volatile-write (ts[i := (p$'$, is$'$,j$'$, sb$'$, $\mathcal{D}'$, $\mathcal{O}'$,$\mathcal{R}'$)]) $\mathcal{S}$
proof (induct ts)
case Nil
thus ?case **by** simp
next
case (Cons t ts)
obtain p_t is_t $\mathcal{O}_t$ $\mathcal{R}_t$ $\mathcal{D}_t$ j_t sb_t **where**
t: t=(p_t,is_t,j_t,sb_t,$\mathcal{D}_t$,$\mathcal{O}_t$,$\mathcal{R}_t$)
by (cases t)

show ?case
proof (cases i)
case 0
with t Cons.prems **have** eqs: p_t=p is_t=is $\mathcal{O}_t$=$\mathcal{O}$ $\mathcal{R}_t$=$\mathcal{R}$ j_t=j sb_t=sb $\mathcal{D}_t$=$\mathcal{D}$
by auto

show ?thesis
by (clarsimp simp add: 0 t eqs congr)
next
case (Suc k)
from Cons.prems Suc **obtain** k-bound: k < length ts **and** ts-k: ts!k = (p, is,j, sb, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$)
by auto
from Cons.hyps [OF k-bound ts-k]
show ?thesis
by (clarsimp simp add: t Suc)
qed
qed

lemma share-all-until-volatile-write-append-Ghost$_{sb}$$'$:
assumes out-VWrite$_{sb}$: outstanding-refs is-volatile-Write$_{sb}$ sb $\neq$ {}
assumes i-bound: i < length ts
assumes ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
shows share-all-until-volatile-write ts $\mathcal{S}$ =
share-all-until-volatile-write
(ts[i := (p$'$, is$'$,j$'$, sb @ [Ghost$_{sb}$ A L R W], $\mathcal{D}'$, $\mathcal{O}'$,$\mathcal{R}'$)]) $\mathcal{S}$
proof −
from out-VWrite$_{sb}$
have $\bigwedge$S. share (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) (sb @ [Ghost$_{sb}$ A L R W])) S =
share (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) S
by (simp add: outstanding-vol-write-takeWhile-append)
from share-all-until-volatile-write-update-sb [OF this i-bound ts-i]
show ?thesis
by simp

qed

lemma acquired-append-Prog$_{sb}$:
$\bigwedge$S. (acquired pending-write (takeWhile (Not ∘ is-volatile-Write$_{sb}$) (sb @ [Prog$_{sb}$ p_1 p_2 mis])) S) =
(acquired pending-write (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) S)
by (induct sb) (auto split: memref.splits)

lemma outstanding-refs-non-empty-dropWhile:
outstanding-refs P xs ≠ {} ⟹ outstanding-refs P (dropWhile (Not ∘ P) xs) ≠ {}
apply (induct xs)
apply simp
apply (simp split: if-split-asm)
done

lemma ex-not: Ex Not
by blast

lemma (**in** computation) concurrent-step-append:
assumes step: (ts,m,$\mathcal{S}$) ⇒ (ts′,m′,$\mathcal{S}$′)
shows (xs@ts,m,$\mathcal{S}$) ⇒ (xs@ts′,m′,$\mathcal{S}$′)
using step
proof (cases)
case (Program i p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ p′ is′)
then obtain
i-bound: i < length ts **and**
ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) **and**
prog-step: j⊢p $\rightarrow_p$ (p′,is′) **and**
ts′: ts′=ts[i:=(p′,is@is′,j,record p p′ is′ sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] **and**
$\mathcal{S}$′: $\mathcal{S}$′=$\mathcal{S}$ **and**
m′: m′=m
by auto

from i-bound **have** i-bound′: i + length xs < length (xs@ts)
by auto

from ts-i i-bound **have** ts-i′: (xs@ts)!(i + length xs) = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by (auto simp add: nth-append)

from concurrent-step.Program [OF i-bound′ ts-i′ prog-step, of m $\mathcal{S}$] ts′ i-bound
show ?thesis
by (auto simp add: ts′ list-update-append $\mathcal{S}'$ m′)
next
case (Memop i p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ is′ j′ sb′ $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$)
then obtain
i-bound: i < length ts **and**
ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) **and**
memop-step: (is,j,sb,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{m}}$ (is′,j′,sb′,m′,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$) **and**
ts′: ts′=ts[i:=(p,is′,j′,sb′,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)]
by auto

from i-bound **have** i-bound′: i + length xs < length (xs@ts)
by auto

from ts-i i-bound **have** ts-i′: (xs@ts)!(i + length xs) = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by (auto simp add: nth-append)

from concurrent-step.Memop [OF i-bound′ ts-i′ memop-step] ts′ i-bound
show ?thesis
by (auto simp add: ts′ list-update-append)
next
case (StoreBuffer i p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ sb′ $\mathcal{O}'$ $\mathcal{R}'$)
then obtain
i-bound: i < length ts **and**
ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) **and**
sb-step: (m,sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{sb}}$ (m′,sb′,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$) **and**
ts′: ts′=ts[i:=(p,is,j,sb′,$\mathcal{D}$,$\mathcal{O}'$,$\mathcal{R}'$)]
by auto
from i-bound **have** i-bound′: i + length xs < length (xs@ts)
by auto

from ts-i i-bound **have** ts-i′: (xs@ts)!(i + length xs) = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by (auto simp add: nth-append)

from concurrent-step.StoreBuffer [OF i-bound′ ts-i′ sb-step] ts′ i-bound
show ?thesis
by (auto simp add: ts′ list-update-append)
qed

primrec weak-sharing-consistent:: owns ⇒ ′a memref list ⇒ bool
where
weak-sharing-consistent $\mathcal{O}$ [] = True
| weak-sharing-consistent $\mathcal{O}$ (r#rs) =
(case r of
$\mathrm{Write}_{\mathsf{sb}}$ volatile - - - A L R W ⇒
(if volatile then L ⊆ A ∧ A ∩ R = {} ∧ R ⊆ $\mathcal{O}$ ∧
weak-sharing-consistent ($\mathcal{O}$ ∪ A − R) rs
else weak-sharing-consistent $\mathcal{O}$ rs)

| $\text{Ghost}_{\mathsf{sb}}$ A L R W ⇒ L ⊆ A ∧ A ∩ R = {} ∧ R ⊆ $\mathcal{O}$ ∧ weak-sharing-consistent ($\mathcal{O}$ ∪ A − R) rs
| - ⇒ weak-sharing-consistent $\mathcal{O}$ rs)

lemma sharing-consistent-weak-sharing-consistent:
$\bigwedge\mathcal{S}\ \mathcal{O}$. sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb $\Longrightarrow$ weak-sharing-consistent $\mathcal{O}$ sb
apply (induct sb)
apply (auto split: memref.splits)
done

lemma weak-sharing-consistent-append:
$\bigwedge\mathcal{O}$. weak-sharing-consistent $\mathcal{O}$ (xs @ ys) =
(weak-sharing-consistent $\mathcal{O}$ xs ∧ weak-sharing-consistent (acquired True xs $\mathcal{O}$) ys)
apply (induct xs)
apply (auto split: memref.splits)
done

lemma read-only-share-unowned: $\bigwedge\mathcal{O}\ \mathcal{S}$.
⟦weak-sharing-consistent $\mathcal{O}$ sb; a ∉ $\mathcal{O}$ ∪ all-acquired sb; a ∈ read-only (share sb $\mathcal{S}$)⟧
$\Longrightarrow$ a ∈ read-only $\mathcal{S}$
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a′ sop v A L R W)
show ?thesis
proof (cases volatile)
case False
with Cons $\text{Write}_{\mathsf{sb}}$ **show** ?thesis **by** auto
next
case True
from Cons.hyps [**where** $\mathcal{S}$=($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) **and** $\mathcal{O}$=($\mathcal{O}$ ∪ A − R)] Cons.prems
show ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ True in-read-only-restrict-conv in-read-only-augment-conv split: if-split-asm)
qed
next
case $\text{Read}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** auto
next
case $\text{Prog}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** auto
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
with Cons.hyps [**where** $\mathcal{S}$=($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) **and** $\mathcal{O}$=($\mathcal{O}$ ∪ A − R)] Cons.prems **show** ?thesis
by (auto simp add: in-read-only-restrict-conv in-read-only-augment-conv split: if-split-asm)
qed
qed

lemma share-read-only-mono-in:
assumes a-in: a $\in$ read-only (share sb $\mathcal{S}$)
assumes ss: read-only $\mathcal{S} \subseteq$ read-only $\mathcal{S}'$
shows a $\in$ read-only (share sb $\mathcal{S}'$)
using share-read-only-mono [OF ss] a-in
by auto

lemma read-only-unacquired-share:
$\bigwedge$S $\mathcal{O}$. $[\![\mathcal{O} \cap$ read-only S = {}; weak-sharing-consistent $\mathcal{O}$ sb; a $\in$ read-only S;
a $\notin$ all-acquired sb $]\!]$
$\Longrightarrow$ a $\in$ read-only (share sb S)
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case (Write$_{sb}$ volatile a$'$ sop v A L R W)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems
obtain a-ro: a $\in$ read-only S **and** a-A: a $\notin$ A **and** a-unacq: a $\notin$ all-acquired sb **and**
owns-ro: $\mathcal{O} \cap$ read-only S = {} **and**
L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq \mathcal{O}$ **and**
consis$'$: weak-sharing-consistent ($\mathcal{O} \cup$ A $-$ R) sb
by (clarsimp simp add: Write$_{sb}$ True)

from owns-ro A-R R-owns **have** owns-ro$'$: ($\mathcal{O} \cup$ A $-$ R) $\cap$ read-only (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
= {}
by (auto simp add: in-read-only-convs)

from a-ro a-A owns-ro R-owns L-A **have** a-ro$'$: a $\in$ read-only (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (auto simp add: in-read-only-convs)
from Cons.hyps [OF owns-ro$'$ consis$'$ a-ro$'$ a-unacq]
show ?thesis
by (clarsimp simp add: Write$_{sb}$ True)
next
case False
with Cons **show** ?thesis
by (clarsimp simp add: Write$_{sb}$ False)

qed
next
case $\text{Read}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** (clarsimp)
next
case $\text{Prog}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** (clarsimp)
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.prems
obtain a-ro: a $\in$ read-only S **and** a-A: a $\notin$ A **and** a-unacq: a $\notin$ all-acquired sb **and**
owns-ro: $\mathcal{O}$ $\cap$ read-only S = {} **and**
L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and**
consis′: weak-sharing-consistent ($\mathcal{O}$ $\cup$ A − R) sb
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)

from owns-ro A-R R-owns **have** owns-ro′: ($\mathcal{O}$ $\cup$ A − R) $\cap$ read-only (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = {}
by (auto simp add: in-read-only-convs)

from a-ro a-A owns-ro R-owns L-A **have** a-ro′: a $\in$ read-only (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (auto simp add: in-read-only-convs)
from Cons.hyps [OF owns-ro′ consis′ a-ro′ a-unacq]
show ?thesis
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
qed
qed

lemma read-only-share-unacquired: $\bigwedge$ $\mathcal{O}$ S. $\mathcal{O}$ $\cap$ read-only S = {} $\Longrightarrow$
weak-sharing-consistent $\mathcal{O}$ sb $\Longrightarrow$
a $\in$ read-only (share sb S) $\Longrightarrow$ a $\notin$ acquired True sb $\mathcal{O}$
proof (induct sb)
case Nil **thus** ?case **by** auto
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a′ sop v A L R W)
show ?thesis
proof (cases volatile)
case False
with Cons $\text{Write}_{\mathsf{sb}}$ **show** ?thesis **by** auto
next
case True
note volatile=this
from Cons.prems
obtain a-ro: a $\in$ read-only (share sb (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) **and**
owns-ro: $\mathcal{O}$ $\cap$ read-only S = {} **and**
L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and**
consis′: weak-sharing-consistent ($\mathcal{O}$ $\cup$ A − R) sb

by (clarsimp simp add: $\mathsf{Write}_{\mathsf{sb}}$ volatile)

from owns-ro A-R R-owns **have** owns-ro′: ($\mathcal{O}$ ∪ A − R) ∩ read-only (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = {}
by (auto simp add: in-read-only-convs)
from Cons.hyps [OF owns-ro′ consis′ a-ro]
show ?thesis
by (auto simp add: $\mathsf{Write}_{\mathsf{sb}}$ volatile)
qed
next
case $\mathsf{Read}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** auto
next
case $\mathsf{Prog}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** auto
next
case ($\mathsf{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.prems
obtain a-ro: a ∈ read-only (share sb (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) **and**
owns-ro: $\mathcal{O}$ ∩ read-only S = {} **and**
L-A: L ⊆ A **and** A-R: A ∩ R = {} **and** R-owns: R ⊆ $\mathcal{O}$ **and**
consis′: weak-sharing-consistent ($\mathcal{O}$ ∪ A − R) sb
by (clarsimp simp add: $\mathsf{Ghost}_{\mathsf{sb}}$)

from owns-ro A-R R-owns **have** owns-ro′: ($\mathcal{O}$ ∪ A − R) ∩ read-only (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = {}
by (auto simp add: in-read-only-convs)
from Cons.hyps [OF owns-ro′ consis′ a-ro]
show ?thesis
by (auto simp add: $\mathsf{Ghost}_{\mathsf{sb}}$)
qed
qed

lemma read-only-share-all-acquired-in:
⋀S $\mathcal{O}$. ⟦$\mathcal{O}$ ∩ read-only S = {}; weak-sharing-consistent $\mathcal{O}$ sb; a ∈ read-only (share sb S)⟧

⟹ a ∈ read-only (share sb Map.empty) ∨ (a ∈ read-only S ∧ a ∉ all-acquired sb)
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\mathsf{Write}_{\mathsf{sb}}$ volatile a′ sop v A L R W)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems
obtain a-in: a ∈ read-only (share sb (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) **and**
owns-ro: $\mathcal{O}$ ∩ read-only S = {} **and**

L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and**
consis′: weak-sharing-consistent ($\mathcal{O}$ $\cup$ A $-$ R) sb
by (clarsimp simp add: Write$_{\mathsf{sb}}$ True)

from owns-ro A-R R-owns **have** owns-ro′: ($\mathcal{O}$ $\cup$ A $-$ R) $\cap$ read-only (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = {}
by (auto simp add: in-read-only-convs)

from Cons.hyps [OF owns-ro′ consis′ a-in]
have hyp: a $\in$ read-only (share sb Map.empty) $\vee$ a $\in$ read-only (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $\wedge$ a $\notin$ all-acquired sb**.**

have a $\in$ read-only (share sb (Map.empty $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) $\vee$ (a $\in$ read-only S $\wedge$ a $\notin$ A $\wedge$ a $\notin$ all-acquired sb)
proof $-$
{
assume a-emp: a $\in$ read-only (share sb Map.empty)
have read-only Map.empty $\subseteq$ read-only (Map.empty $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (auto simp add: in-read-only-convs)

from share-read-only-mono-in [OF a-emp this]
have a $\in$ read-only (share sb (Map.empty $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L))**.**
}
moreover
{
assume a-ro: a $\in$ read-only (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) **and** a-unacq: a $\notin$ all-acquired sb
have ?thesis
proof (cases a $\in$ read-only S)
case True
with a-ro **obtain** a $\notin$ A
by (auto simp add: in-read-only-convs)
with True a-unacq **show** ?thesis
by auto
next
case False
with a-ro **have** a-ro-empty: a $\in$ read-only (Map.empty $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (auto simp add: in-read-only-convs split: if-split-asm)

have read-only (Map.empty $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $\subseteq$ read-only (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (auto simp add: in-read-only-convs)
with owns-ro′
have owns-ro-empty: ($\mathcal{O}$ $\cup$ A $-$ R) $\cap$ read-only (Map.empty $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = {}
by blast

from read-only-unacquired-share [OF owns-ro-empty consis′ a-ro-empty a-unacq]
have a $\in$ read-only (share sb (Map.empty $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L))**.**
thus ?thesis
by simp
qed

}
moreover note hyp
ultimately show ?thesis **by** blast
qed

then show ?thesis
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ True)
next
case False **with** Cons **show** ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$)
qed
next
case $\text{Read}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** auto
next
case $\text{Prog}_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** auto
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.prems
obtain a-in: a $\in$ read-only (share sb (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) **and**
owns-ro: $\mathcal{O}$ $\cap$ read-only S = {} **and**
L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and**
consis$'$: weak-sharing-consistent ($\mathcal{O}$ $\cup$ A $-$ R) sb
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)

from owns-ro A-R R-owns **have** owns-ro$'$: ($\mathcal{O}$ $\cup$ A $-$ R) $\cap$ read-only (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = {}
by (auto simp add: in-read-only-convs)

from Cons.hyps [OF owns-ro$'$ consis$'$ a-in]
have hyp: a $\in$ read-only (share sb Map.empty) $\vee$ a $\in$ read-only (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $\wedge$ a $\notin$ all-acquired sb**.**

have a $\in$ read-only (share sb (Map.empty $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) $\vee$ (a $\in$ read-only S $\wedge$ a $\notin$ A $\wedge$ a $\notin$ all-acquired sb)
proof $-$
{
assume a-emp: a $\in$ read-only (share sb Map.empty)
have read-only Map.empty $\subseteq$ read-only (Map.empty $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (auto simp add: in-read-only-convs)

from share-read-only-mono-in [OF a-emp this]
have a $\in$ read-only (share sb (Map.empty $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L))**.**
}
moreover
{
assume a-ro: a $\in$ read-only (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) **and** a-unacq: a $\notin$ all-acquired sb
have ?thesis
proof (cases a $\in$ read-only S)
case True
with a-ro **obtain** a $\notin$ A

by (auto simp add: in-read-only-convs)
with True a-unacq **show** ?thesis
by auto
next
case False
with a-ro **have** a-ro-empty: a ∈ read-only (Map.empty $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (auto simp add: in-read-only-convs split: if-split-asm)

have read-only (Map.empty $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ⊆ read-only (S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (auto simp add: in-read-only-convs)
with owns-ro′
have owns-ro-empty: ($\mathcal{O}$ ∪ A − R) ∩ read-only (Map.empty $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = {}
by blast

from read-only-unacquired-share [OF owns-ro-empty consis′ a-ro-empty a-unacq]
have a ∈ read-only (share sb (Map.empty $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L))**.**
thus ?thesis
by simp
qed
}
moreover note hyp
ultimately show ?thesis **by** blast
qed
then show ?thesis
by (clarsimp simp add: $\mathrm{Ghost}_{\mathsf{sb}}$)
qed
qed

lemma weak-sharing-consistent-preserves-distinct:
$\bigwedge\mathcal{O}\ \mathcal{S}$. weak-sharing-consistent $\mathcal{O}$ sb ⟹ $\mathcal{O}$ ∩ read-only $\mathcal{S}$ = {} ⟹
acquired True sb $\mathcal{O}$ ∩ read-only (share sb $\mathcal{S}$) = {}
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\mathrm{Write}_{\mathsf{sb}}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems **obtain**
owns-ro: $\mathcal{O}$ ∩ read-only $\mathcal{S}$ = {} **and** L-A: L ⊆ A **and** A-R: A ∩ R = {} **and**
R-owns: R ⊆ $\mathcal{O}$ **and** consis′: weak-sharing-consistent ($\mathcal{O}$ ∪ A − R) sb
by (clarsimp simp add: $\mathrm{Write}_{\mathsf{sb}}$ True)

from owns-ro A-R R-owns **have** owns-ro′: ($\mathcal{O}$ ∪ A − R) ∩ read-only ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = {}
by (auto simp add: in-read-only-convs)
from Cons.hyps [OF consis′ owns-ro′]
show ?thesis
by (auto simp add: Write$_{\mathsf{sb}}$ True)
next
case False **with** Cons Write$_{\mathsf{sb}}$ **show** ?thesis **by** auto
qed
next
case Read$_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** auto
next
case Prog$_{\mathsf{sb}}$ **with** Cons **show** ?thesis **by** auto
next
case (Ghost$_{\mathsf{sb}}$ A L R W)
from Cons.prems **obtain**
owns-ro: $\mathcal{O}$ ∩ read-only $\mathcal{S}$ = {} **and** L-A: L ⊆ A **and** A-R: A ∩ R = {} **and**
R-owns: R ⊆ $\mathcal{O}$ **and** consis′: weak-sharing-consistent ($\mathcal{O}$ ∪ A − R) sb
by (clarsimp simp add: Ghost$_{\mathsf{sb}}$)

from owns-ro A-R R-owns **have** owns-ro′: ($\mathcal{O}$ ∪ A − R) ∩ read-only ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = {}
by (auto simp add: in-read-only-convs)
from Cons.hyps [OF consis′ owns-ro′]
show ?thesis
by (auto simp add: Ghost$_{\mathsf{sb}}$)
qed
qed

locale weak-sharing-consis =
fixes ts::(′p,′p store-buffer,bool,owns,rels) thread-config list
assumes weak-sharing-consis:
⋀i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
⟦i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$) ⟧
⟹
weak-sharing-consistent $\mathcal{O}$ sb

sublocale sharing-consis ⊆ weak-sharing-consis
proof
fix i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
assume i-bound: i < length ts
assume ts-i: ts ! i = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)
from sharing-consistent-weak-sharing-consistent [OF sharing-consis [OF i-bound ts-i]]
show weak-sharing-consistent $\mathcal{O}$ sb**.**
qed

lemma weak-sharing-consis-tl: weak-sharing-consis (t#ts) ⟹ weak-sharing-consis ts

apply (unfold weak-sharing-consis-def)
apply force
done

lemma read-only-share-all-until-volatile-write-unacquired:
  $\bigwedge\mathcal{S}$. ⟦ownership-distinct ts; read-only-unowned $\mathcal{S}$ ts; weak-sharing-consis ts;
  $\forall$i < length ts. (let (-,-,-,sb,-,$\mathcal{O}$,$\mathcal{R}$) = ts!i in
    a $\notin$ all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb));
  a $\in$ read-only $\mathcal{S}$⟧
  $\Longrightarrow$ a $\in$ read-only (share-all-until-volatile-write ts $\mathcal{S}$)
proof (induct ts)
  case Nil **thus** ?case **by** simp
next
  case (Cons t ts)
  obtain p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb **where**
    t: t = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
    by (cases t)

  have dist: ownership-distinct (t#ts) **by** fact
  then interpret ownership-distinct t#ts **.**
  from ownership-distinct-tl [OF dist]
  have dist′: ownership-distinct ts**.**

  have aargh: (Not $\circ$ is-volatile-Write$_{sb}$) = (λa. ¬ is-volatile-Write$_{sb}$ a)
    by (rule ext) auto

  have a-ro: a $\in$ read-only $\mathcal{S}$ **by** fact
  have ro-unowned: read-only-unowned $\mathcal{S}$ (t#ts) **by** fact
  then interpret read-only-unowned $\mathcal{S}$ t#ts **.**
  have consis: weak-sharing-consis (t#ts) **by** fact
  then interpret weak-sharing-consis t#ts **.**

  note consis′ = weak-sharing-consis-tl [OF consis]

  let ?take-sb = (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb)
  let ?drop-sb = (dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb)

  from weak-sharing-consis [of 0] t
  have consis-sb: weak-sharing-consistent $\mathcal{O}$ sb
    by force
  with weak-sharing-consistent-append [of $\mathcal{O}$?take-sb ?drop-sb]
  have consis-take: weak-sharing-consistent $\mathcal{O}$?take-sb
    by auto

  have ro-unowned′: read-only-unowned (share ?take-sb $\mathcal{S}$) ts
  proof
    fix j

fix p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j
assume j-bound: j < length ts
assume jth: ts!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show $\mathcal{O}_j$ ∩ read-only (share ?take-sb $\mathcal{S}$) = {}
proof −
{
fix a
assume a-owns: a ∈ $\mathcal{O}_j$
assume a-ro: a ∈ read-only (share ?take-sb $\mathcal{S}$)
have False
proof −
from ownership-distinct [of 0 Suc j] j-bound jth t
have dist: ($\mathcal{O}$ ∪ all-acquired sb) ∩ ($\mathcal{O}_j$ ∪ all-acquired sb_j) = {}
by fastforce

from read-only-unowned [of Suc j] j-bound jth
have dist-ro: $\mathcal{O}_j$ ∩ read-only $\mathcal{S}$ = {} **by** force
show ?thesis
proof (cases a ∈ ($\mathcal{O}$ ∪ all-acquired sb))
case True
with dist a-owns **show** False **by** auto
next
case False
hence a ∉ ($\mathcal{O}$ ∪ all-acquired ?take-sb)
using all-acquired-append [of ?take-sb ?drop-sb]
by auto
from read-only-share-unowned [OF consis-take this a-ro]
have a ∈ read-only $\mathcal{S}$**.**
with dist-ro a-owns **show** False **by** auto
qed
qed
}
thus ?thesis **by** auto
qed
qed

from Cons.prems
obtain unacq-ts: ∀ i < length ts. (let (-,-,-,sb,-,$\mathcal{O}$,-) = ts!i in
a ∉ all-acquired (takeWhile (Not ∘ is-volatile-Write_{sb}) sb)) **and**
unacq-sb: a ∉ all-acquired (takeWhile (Not ∘ is-volatile-Write_{sb}) sb)
by (force simp add: t aargh)

from read-only-unowned [of 0] t
have owns-ro: $\mathcal{O}$ ∩ read-only $\mathcal{S}$ = {}
by force
from read-only-unacquired-share [OF owns-ro consis-take a-ro unacq-sb]
have a ∈ read-only (share (takeWhile (Not ∘ is-volatile-Write_{sb}) sb) $\mathcal{S}$)**.**

from Cons.hyps [OF dist′ ro-unowned′ consis′ unacq-ts this]
show ?case
by (simp add: t)
qed

lemma read-only-share-unowned-in:
⟦weak-sharing-consistent $\mathcal{O}$ sb; a ∈ read-only (share sb $\mathcal{S}$)⟧
⟹ a ∈ read-only $\mathcal{S}$ ∪ $\mathcal{O}$ ∪ all-acquired sb
using read-only-share-unowned [of $\mathcal{O}$ sb]
by auto

lemma read-only-shared-all-until-volatile-write-subset:
⋀$\mathcal{S}$. ⟦ownership-distinct ts;
weak-sharing-consis ts⟧ ⟹
read-only (share-all-until-volatile-write ts $\mathcal{S}$) ⊆
read-only $\mathcal{S}$ ∪ (⋃((λ(-, -, -, sb, -, $\mathcal{O}$,-). $\mathcal{O}$ ∪ all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)) ' set ts))
proof (induct ts)
case Nil **thus** ?case **by** simp
next
case (Cons t ts)
obtain p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb **where**
t: t = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
by (cases t)

have dist: ownership-distinct (t#ts) **by** fact
then interpret ownership-distinct t#ts **.**
from ownership-distinct-tl [OF dist]
have dist′: ownership-distinct ts**.**

have consis: weak-sharing-consis (t#ts) **by** fact
then interpret weak-sharing-consis t#ts **.**

have aargh: (Not ∘ is-volatile-Write$_{sb}$) = (λa. ¬ is-volatile-Write$_{sb}$ a)
by (rule ext) auto

note consis′ = weak-sharing-consis-tl [OF consis]

let ?take-sb = (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
let ?drop-sb = (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb)

from weak-sharing-consis [of 0] t
have consis-sb: weak-sharing-consistent $\mathcal{O}$ sb
by force
with weak-sharing-consistent-append [of $\mathcal{O}$?take-sb ?drop-sb]
have consis-take: weak-sharing-consistent $\mathcal{O}$?take-sb
by auto

{
fix a
assume a-in: a ∈ read-only
(share-all-until-volatile-write ts
(share ?take-sb $\mathcal{S}$)) **and**
a-notin-shared: a ∉ read-only $\mathcal{S}$ **and**
a-notin-rest: a ∉ (⋃((λ(-, -, -, sb, -, $\mathcal{O}$,-). $\mathcal{O}$ ∪ all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)) ‘ set ts))
have a ∈ $\mathcal{O}$ ∪ all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
proof −
from Cons.hyps [OF dist′ consis′, of (share ?take-sb $\mathcal{S}$)] a-in a-notin-rest
have a ∈ read-only (share ?take-sb $\mathcal{S}$)
by (auto simp add: aargh)
from read-only-share-unowned-in [OF consis-take this] a-notin-shared
show ?thesis **by** auto
qed
}

then show ?case
by (auto simp add: t aargh)
qed

lemma weak-sharing-consistent-preserves-distinct-share-all-until-volatile-write:
⋀$\mathcal{S}$ i. ⟦ownership-distinct ts; read-only-unowned $\mathcal{S}$ ts;weak-sharing-consis ts;
i < length ts; ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)⟧
⟹ acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{O}$ ∩
read-only (share-all-until-volatile-write ts $\mathcal{S}$) = {}
proof (induct ts)
case Nil **thus** ?case **by** simp
next
case (Cons t ts)
note ‹read-only-unowned $\mathcal{S}$ (t#ts)›
then interpret read-only-unowned $\mathcal{S}$ t#ts **.**
note i-bound = ‹i < length (t # ts)›
note ith = ‹(t # ts) ! i = (p,is,j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)›

have dist: ownership-distinct (t#ts) **by** fact
then interpret ownership-distinct t#ts **.**
from ownership-distinct-tl [OF dist]
have dist′: ownership-distinct ts**.**

have consis: weak-sharing-consis (t#ts) **by** fact
then interpret weak-sharing-consis t#ts **.**

note consis′ = weak-sharing-consis-tl [OF consis]

let ?take-sb = (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
let ?drop-sb = (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb)

have aargh: (Not ∘ is-volatile-Write$_{sb}$) = (λa. ¬ is-volatile-Write$_{sb}$ a)
by (rule ext) auto
show ?case
proof (cases i)
case 0
from read-only-unowned [of 0] ith 0
have owns-ro: $\mathcal{O}$ ∩ read-only $\mathcal{S}$ = {}
by force
from weak-sharing-consis [of 0] ith 0
have weak-sharing-consistent $\mathcal{O}$ sb
by force
with weak-sharing-consistent-append [of $\mathcal{O}$?take-sb ?drop-sb]
have consis-take: weak-sharing-consistent $\mathcal{O}$?take-sb
by auto
from weak-sharing-consistent-preserves-distinct [OF this owns-ro]
have dist-t: acquired True ?take-sb $\mathcal{O}$ ∩ read-only (share ?take-sb $\mathcal{S}$) = {}.
from read-only-shared-all-until-volatile-write-subset [OF dist′ consis′, of (share ?take-sb $\mathcal{S}$)]
have ro-rest: read-only (share-all-until-volatile-write ts (share ?take-sb $\mathcal{S}$)) ⊆
read-only (share ?take-sb $\mathcal{S}$) ∪
($\bigcup$((λ(-, -, -, sb, -, $\mathcal{O}$,-). $\mathcal{O}$ ∪ all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)) ‘ set ts))
by auto
{
fix a
assume a-in-sb: a ∈ acquired True ?take-sb $\mathcal{O}$
assume a-in-ro: a ∈ read-only (share-all-until-volatile-write ts (share ?take-sb $\mathcal{S}$))
have False
proof −

from Set.in-mono [rule-format, OF ro-rest a-in-ro] dist-t a-in-sb

have a ∈ ($\bigcup$((λ(-, -, -, sb, -, $\mathcal{O}$,-). $\mathcal{O}$ ∪ all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)) ‘ set ts))
by auto
then obtain j p_j is_j j_j sb_j $\mathcal{D}_j$ $\mathcal{O}_j$ $\mathcal{R}_j$
where j-bound: j < length ts **and** ts-j: ts!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
and a-in-j: a ∈ $\mathcal{O}_j$ ∪ all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)
by (fastforce simp add: in-set-conv-nth)
from ownership-distinct [of 0 Suc j] ith ts-j j-bound 0
have dist: ($\mathcal{O}$ ∪ all-acquired sb) ∩ ($\mathcal{O}_j$ ∪ all-acquired sb_j) = {}
by fastforce
moreover
from acquired-all-acquired [of True ?take-sb $\mathcal{O}$] a-in-sb all-acquired-append [of ?take-sb ?drop-sb]
have a ∈ $\mathcal{O}$ ∪ all-acquired sb
by auto
with a-in-j all-acquired-append [of (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j) (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)]

```
        dist
      have False by fastforce
      thus ?thesis ..
    qed
  }
  then show ?thesis
  using 0 ith
    by (auto simp add: aargh)
 next
   case (Suc k)
   from i-bound Suc have k-bound: k < length ts
     by auto
   from ith Suc have kth: ts!k = (p, is, j, sb, 𝒟, 𝒪, ℛ)
     by auto

   obtain pₜ isₜ 𝒪ₜ ℛₜ 𝒟ₜ jₜ sbₜ
     where t: t=(pₜ,isₜ,jₜ,sbₜ,𝒟ₜ,𝒪ₜ,ℛₜ)
     by (cases t)

   let ?take-sbₜ = (takeWhile (Not ∘ is-volatile-Write_sb) sbₜ)
   let ?drop-sbₜ = (dropWhile (Not ∘ is-volatile-Write_sb) sbₜ)

   have ro-unowned': read-only-unowned (share ?take-sbₜ 𝒮) ts
   proof
     fix j
     fix pⱼ isⱼ 𝒪ⱼ ℛⱼ 𝒟ⱼ jⱼ sbⱼ
     assume j-bound: j < length ts
     assume jth: ts!j = (pⱼ,isⱼ,jⱼ,sbⱼ,𝒟ⱼ,𝒪ⱼ,ℛⱼ)
     show 𝒪ⱼ ∩ read-only (share ?take-sbₜ 𝒮) = {}
     proof -
from read-only-unowned [of Suc j] j-bound jth
have dist: 𝒪ⱼ ∩ read-only 𝒮 = {} by force

      from weak-sharing-consis [of 0] t
      have weak-sharing-consistent 𝒪ₜ sbₜ
        by fastforce
      with weak-sharing-consistent-append [of 𝒪ₜ ?take-sbₜ ?drop-sbₜ]
      have consis-t: weak-sharing-consistent 𝒪ₜ ?take-sbₜ
        by auto
      {
        fix a
        assume a-in-j: a ∈ 𝒪ⱼ
        assume a-ro: a ∈ read-only (share ?take-sbₜ 𝒮)
        have False
        proof -
          from a-in-j ownership-distinct [of 0 Suc j] j-bound t jth
          have (𝒪ₜ ∪ all-acquired sbₜ) ∩ (𝒪ⱼ ∪ all-acquired sbⱼ) = {}
            by fastforce
          with a-in-j all-acquired-append [of ?take-sbₜ ?drop-sbₜ]
          have a ∉ (𝒪ₜ ∪ all-acquired ?take-sbₜ)
```

```
          by fastforce
        from read-only-share-unowned [OF consis-t this a-ro]
        have a ∈ read-only 𝒮 .
        with a-in-j dist
        show False by auto
      qed
    }
    then
 show ?thesis
  by auto
   qed
  qed

  from Cons.hyps [OF dist′ ro-unowned′ consis′ k-bound kth]
  show ?thesis
   by (simp add: t)
 qed
qed


lemma in-read-only-share-all-until-volatile-write:
 assumes dist: ownership-distinct ts
 assumes consis: sharing-consis 𝒮 ts
 assumes ro-unowned: read-only-unowned 𝒮 ts
 assumes i-bound: i < length ts
 assumes ts-i: ts!i = (p,is,j,sb,𝒟,𝒪,ℛ)
 assumes a-unacquired-others: ∀j < length ts. i≠j ⟶
     (let (-,-,-,sb_j,-,-,-) = ts!j in
     a ∉ all-acquired (takeWhile (Not ∘ is-volatile-Write_sb) sb_j))
 assumes a-ro-share: a ∈ read-only (share sb 𝒮)
 shows a ∈ read-only (share (dropWhile (Not ∘ is-volatile-Write_sb) sb)
         (share-all-until-volatile-write ts 𝒮))
proof -
 from consis
 interpret sharing-consis 𝒮 ts .
 interpret read-only-unowned 𝒮 ts by fact

 from sharing-consis [OF i-bound ts-i]
 have consis-sb: sharing-consistent 𝒮 𝒪 sb.
 from sharing-consistent-weak-sharing-consistent [OF this]
 have weak-consis: weak-sharing-consistent 𝒪 sb.
 from read-only-unowned [OF i-bound ts-i]
 have owns-ro: 𝒪 ∩ read-only 𝒮 = {}.
 from read-only-share-all-acquired-in [OF owns-ro weak-consis a-ro-share]
 have a ∈ read-only (share sb Map.empty) ∨ a ∈ read-only 𝒮 ∧ a ∉ all-acquired sb.
 moreover

 let ?take-sb = (takeWhile (Not ∘ is-volatile-Write_sb) sb)
 let ?drop-sb = (dropWhile (Not ∘ is-volatile-Write_sb) sb)
```

from weak-consis weak-sharing-consistent-append [of $\mathcal{O}$?take-sb ?drop-sb]
obtain weak-consis′: weak-sharing-consistent (acquired True ?take-sb $\mathcal{O}$) ?drop-sb **and**
weak-consis-take: weak-sharing-consistent $\mathcal{O}$?take-sb
by auto

{
assume a ∈ read-only (share sb Map.empty)
with share-append [of ?take-sb ?drop-sb]
have a-in′: a ∈ read-only (share ?drop-sb (share ?take-sb Map.empty))
by auto

have owns-empty: $\mathcal{O}$ ∩ read-only Map.empty = {}
by auto

from weak-sharing-consistent-preserves-distinct [OF weak-consis-take owns-empty]
have acquired True ?take-sb $\mathcal{O}$ ∩ read-only (share ?take-sb Map.empty) = {}**.**

from read-only-share-all-acquired-in [OF this weak-consis′ a-in′]
have a ∈ read-only (share ?drop-sb Map.empty) ∨ a ∈ read-only (share ?take-sb Map.empty) ∧ a ∉ all-acquired ?drop-sb**.**
moreover
{
assume a-ro-drop: a ∈ read-only (share ?drop-sb Map.empty)
have read-only Map.empty ⊆ read-only (share-all-until-volatile-write ts $\mathcal{S}$)
by auto
from share-read-only-mono-in [OF a-ro-drop this]
have ?thesis **.**
}
moreover
{
assume a-ro-take: a ∈ read-only (share ?take-sb Map.empty)
assume a-unacq-drop: a ∉ all-acquired ?drop-sb
from read-only-share-unowned-in [OF weak-consis-take a-ro-take]
have a ∈ $\mathcal{O}$ ∪ all-acquired ?take-sb **by** auto
hence a ∈ $\mathcal{O}$ ∪ all-acquired sb **using** all-acquired-append [of ?take-sb ?drop-sb]
by auto
from share-all-until-volatile-write-thread-local′ [OF dist consis i-bound ts-i this] a-ro-share
have ?thesis **by** (auto simp add: read-only-def)
}
ultimately have ?thesis **by** blast
}

moreover

{
assume a-ro: a ∈ read-only $\mathcal{S}$
assume a-unacq: a ∉ all-acquired sb
with all-acquired-append [of ?take-sb ?drop-sb]
obtain a ∉ all-acquired ?take-sb **and** a-notin-drop: a ∉ all-acquired ?drop-sb

by auto
with a-unacquired-others i-bound ts-i
have a-unacq: $\forall$ j < length ts.
(let (-,-,-,sb_j,-,-,-) = ts!j in
a $\notin$ all-acquired (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j))
by (auto simp add: Let-def)

from local.weak-sharing-consis-axioms **have** weak-sharing-consis ts **.**
from read-only-share-all-until-volatile-write-unacquired [OF dist ro-unowned ‹weak-sharing-consis ts› a-unacq a-ro]
have a-ro-all: a $\in$ read-only (share-all-until-volatile-write ts $\mathcal{S}$) **.**

from weak-consis weak-sharing-consistent-append [of $\mathcal{O}$?take-sb ?drop-sb]
have weak-consis-drop: weak-sharing-consistent (acquired True ?take-sb $\mathcal{O}$) ?drop-sb
by auto

from weak-sharing-consistent-preserves-distinct-share-all-until-volatile-write [OF dist ro-unowned ‹weak-sharing-consis ts› i-bound ts-i]
have acquired True ?take-sb $\mathcal{O}$ $\cap$
read-only (share-all-until-volatile-write ts $\mathcal{S}$) = {}**.**

from read-only-unacquired-share [OF this weak-consis-drop a-ro-all a-notin-drop]
have ?thesis **.**
}
ultimately show ?thesis **by** blast
qed

lemma all-acquired-dropWhile-in: x $\in$ all-acquired (dropWhile P sb) $\Longrightarrow$ x $\in$ all-acquired sb
using all-acquired-append [of takeWhile P sb dropWhile P sb]
by auto

lemma all-acquired-takeWhile-in: x $\in$ all-acquired (takeWhile P sb) $\Longrightarrow$ x $\in$ all-acquired sb
using all-acquired-append [of takeWhile P sb dropWhile P sb]
by auto

lemmas all-acquired-takeWhile-dropWhile-in = all-acquired-takeWhile-in all-acquired-dropWhile-in

lemma split-in-read-only-reads:
$\bigwedge\mathcal{O}$. a $\in$ read-only-reads $\mathcal{O}$ xs $\Longrightarrow$
($\exists$ t v ys zs. xs=ys @ $Read_{sb}$ False a t v # zs $\wedge$ a $\notin$ acquired True ys $\mathcal{O}$)
proof (induct xs)
case Nil **thus** ?case **by** simp
next
case (Cons x xs)

have a-in: a ∈ read-only-reads $\mathcal{O}$ (x#xs) **by** fact
show ?case
proof (cases x)
case (Write$_{sb}$ volatile a′ sop v A L R W)
show ?thesis
proof (cases volatile)
case False
from a-in **have** a ∈ read-only-reads $\mathcal{O}$ xs
by (auto simp add: Write$_{sb}$ False)
from Cons.hyps [OF this] **obtain** t v ys zs **where**
xs: xs=ys@Read$_{sb}$ False a t v # zs **and** a-notin: a ∉ acquired True ys $\mathcal{O}$
by auto
with xs a-notin **obtain** x#xs=(x#ys)@Read$_{sb}$ False a t v # zs a ∉ acquired True (x#ys) $\mathcal{O}$
by (simp add: Write$_{sb}$ False)
then show ?thesis
by blast
next
case True
from a-in **have** a ∈ read-only-reads ($\mathcal{O}$ ∪ A − R) xs
by (auto simp add: Write$_{sb}$ True)
from Cons.hyps [OF this] **obtain** t v ys zs **where**
xs: xs=ys@Read$_{sb}$ False a t v # zs **and** a-notin: a ∉ acquired True ys ($\mathcal{O}$ ∪ A − R)
by auto
with xs a-notin **obtain** x#xs=(x#ys)@Read$_{sb}$ False a t v # zs a ∉ acquired True (x#ys) $\mathcal{O}$
by (simp add: Write$_{sb}$ True)
then show ?thesis
by blast
qed
next
case (Read$_{sb}$ volatile a′ t′ v′)
show ?thesis
proof (cases ¬ volatile ∧ a ∉ $\mathcal{O}$ ∧ a′=a)
case True
with Read$_{sb}$ **show** ?thesis
by fastforce
next
case False
with a-in **have** a ∈ read-only-reads $\mathcal{O}$ xs
by (auto simp add: Read$_{sb}$ split: if-split-asm)
from Cons.hyps [OF this] **obtain** t v ys zs **where**
xs: xs=ys@Read$_{sb}$ False a t v # zs **and** a-notin: a ∉ acquired True ys $\mathcal{O}$
by auto
with xs a-notin **obtain** x#xs=(x#ys)@Read$_{sb}$ False a t v # zs a ∉ acquired True (x#ys) $\mathcal{O}$
by (simp add: Read$_{sb}$)
then show ?thesis
by blast
qed

next
case $Prog_{sb}$
with a-in **have** a ∈ read-only-reads $\mathcal{O}$ xs
by (auto)
from Cons.hyps [OF this] **obtain** t v ys zs **where**
xs: xs=ys@$Read_{sb}$ False a t v # zs **and** a-notin: a ∉ acquired True ys $\mathcal{O}$
by auto
with xs a-notin **obtain** x#xs=(x#ys)@$Read_{sb}$ False a t v # zs a ∉ acquired True (x#ys) $\mathcal{O}$
by (simp add: $Prog_{sb}$)
then show ?thesis
by blast
next
case ($Ghost_{sb}$ A L R W)
with a-in **have** a ∈ read-only-reads ($\mathcal{O}$ ∪ A − R) xs
by (auto)
from Cons.hyps [OF this] **obtain** t v ys zs **where**
xs: xs=ys@$Read_{sb}$ False a t v # zs **and** a-notin: a ∉ acquired True ys ($\mathcal{O}$ ∪ A −R)
by auto
with xs a-notin **obtain** x#xs=(x#ys)@$Read_{sb}$ False a t v # zs a ∉ acquired True (x#ys) $\mathcal{O}$
by (simp add: $Ghost_{sb}$)
then show ?thesis
by blast
qed
qed

lemma insert-monoD: W ⊆ W′ ⟹ insert a W ⊆ insert a W′
by blast

primrec unforwarded-non-volatile-reads:: ′a memref list ⇒ addr set ⇒ addr set
where
unforwarded-non-volatile-reads [] W = {}
| unforwarded-non-volatile-reads (x#xs) W =
(case x of
$Read_{sb}$ volatile a - - ⇒ (if a ∉ W ∧ ¬ volatile
then insert a (unforwarded-non-volatile-reads xs W)
else (unforwarded-non-volatile-reads xs W))
| $Write_{sb}$ - a - - - - - - ⇒ unforwarded-non-volatile-reads xs (insert a W)
| - ⇒ unforwarded-non-volatile-reads xs W)

lemma unforwarded-non-volatile-reads-non-volatile-$Read_{sb}$:
⋀W. unforwarded-non-volatile-reads sb W ⊆ outstanding-refs is-non-volatile-$Read_{sb}$ sb
apply (induct sb)
apply (auto split: memref.splits if-split-asm)
done

lemma in-unforwarded-non-volatile-reads-non-volatile-Read$_{sb}$:
 a ∈ unforwarded-non-volatile-reads sb W ⟹ a ∈ outstanding-refs is-non-volatile-Read$_{sb}$ sb
using unforwarded-non-volatile-reads-non-volatile-Read$_{sb}$
by blast

lemma unforwarded-non-volatile-reads-antimono:
 ⋀W W′. W ⊆ W′ ⟹ unforwarded-non-volatile-reads xs W′ ⊆ unforwarded-non-volatile-reads xs W
apply (induct xs)
apply (auto split: memref.splits dest: insert-monoD)
done

lemma unforwarded-non-volatile-reads-antimono-in:
 x ∈ unforwarded-non-volatile-reads xs W′ ⟹ W ⊆ W′
 ⟹ x ∈ unforwarded-non-volatile-reads xs W
 using unforwarded-non-volatile-reads-antimono
 by blast

lemma unforwarded-non-volatile-reads-append: ⋀W. unforwarded-non-volatile-reads (xs@ys) W =
 (unforwarded-non-volatile-reads xs W ∪
 unforwarded-non-volatile-reads ys (W ∪ outstanding-refs is-Write$_{sb}$ xs))
apply (induct xs)
apply clarsimp
apply (auto split: memref.splits)
done

lemma reads-consistent-mem-eq-on-unforwarded-non-volatile-reads:
 assumes mem-eq: ∀ a ∈ A ∪ W. m′ a = m a
 assumes subset: unforwarded-non-volatile-reads sb W ⊆ A
 assumes consis-m: reads-consistent pending-write $\mathcal{O}$ m sb
 shows reads-consistent pending-write $\mathcal{O}$ m′ sb
using mem-eq subset consis-m
proof (induct sb arbitrary: A W m′ m pending-write $\mathcal{O}$)
 case Nil **thus** ?case **by** simp
next
 case (Cons r sb)
 note mem-eq = ‹∀ a ∈ A ∪ W. m′ a = m a›
 note subset = ‹unforwarded-non-volatile-reads (r#sb) W ⊆ A›
 note consis-m = ‹reads-consistent pending-write $\mathcal{O}$ m (r#sb)›

 show ?case
 proof (cases r)
 case (Write$_{sb}$ volatile a sop v A′ L R W′)
 from subset **obtain**
 subset′: unforwarded-non-volatile-reads sb (insert a W) ⊆ A
 by (auto simp add: Write$_{sb}$)
 from mem-eq

have mem-eq′:
∀a′ ∈ (A ∪ (insert a W)). (m′(a:=v)) a′ = (m(a:=v)) a′
by (auto)
show ?thesis
proof (cases volatile)
case True
from consis-m **obtain**
consis′: reads-consistent True ($\mathcal{O}$ ∪ A′ − R) (m(a := v)) sb **and**
no-volatile-Read$_{sb}$: outstanding-refs is-volatile-Read$_{sb}$ sb = {}
by (simp add: Write$_{sb}$ True)

from Cons.hyps [OF mem-eq′ subset′ consis′]
have reads-consistent True ($\mathcal{O}$ ∪ A′ − R) (m′(a := v)) sb**.**
with no-volatile-Read$_{sb}$
show ?thesis
by (simp add: Write$_{sb}$ True)
next
case False
from consis-m **obtain** consis′: reads-consistent pending-write $\mathcal{O}$ (m(a := v)) sb
by (simp add: Write$_{sb}$ False)
from Cons.hyps [OF mem-eq′ subset′ consis′]
have reads-consistent pending-write $\mathcal{O}$ (m′(a := v)) sb**.**
then
show ?thesis
by (simp add: Write$_{sb}$ False)
qed
next
case (Read$_{sb}$ volatile a t v)
from mem-eq
have mem-eq′:
∀a′ ∈ A ∪ W. m′ a′ = m a′
by (auto)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from consis-m **obtain**
consis′: reads-consistent pending-write $\mathcal{O}$ m sb
by (simp add: Read$_{sb}$ True)

show ?thesis
proof (cases a ∈ W)
case False
from subset **obtain**
subset′: unforwarded-non-volatile-reads sb W ⊆ A
using False
by (auto simp add: Read$_{sb}$ True split: if-split-asm)
from Cons.hyps [OF mem-eq′ subset′ consis′]
show ?thesis
by (simp add: Read$_{sb}$ True)

next
case True
from subset **have**
subset′: unforwarded-non-volatile-reads sb W $\subseteq$
insert a A
using True
apply (auto simp add: $\mathrm{Read}_{\mathsf{sb}}$ volatile split: if-split-asm)
done
from mem-eq True **have** mem-eq′: $\forall$ a′ $\in$ (insert a A) $\cup$ W. m′ a′ = m a′
by auto
from Cons.hyps [OF mem-eq′ subset′ consis′]
show ?thesis
by (simp add: $\mathrm{Read}_{\mathsf{sb}}$ volatile)
qed
next
case False
note non-vol = this
from consis-m **obtain**
consis′: reads-consistent pending-write $\mathcal{O}$ m sb **and**
v: (pending-write $\vee$ a $\in$ $\mathcal{O}$) $\longrightarrow$ v=m a
by (simp add: $\mathrm{Read}_{\mathsf{sb}}$ False)
show ?thesis
proof (cases a $\in$ W)
case True
from mem-eq subset $\mathrm{Read}_{\mathsf{sb}}$ True non-vol **have** m′ a = m a
by (auto simp add: False)
from subset **obtain**
subset′: unforwarded-non-volatile-reads sb W $\subseteq$ insert a A
using False
by (auto simp add: $\mathrm{Read}_{\mathsf{sb}}$ non-vol split: if-split-asm)
from mem-eq True **have** mem-eq′: $\forall$ a′ $\in$ (insert a A) $\cup$ W. m′ a′ = m a′
by auto

with Cons.hyps [OF mem-eq′ subset′ consis′] v
show ?thesis
by (simp add: $\mathrm{Read}_{\mathsf{sb}}$ non-vol)
next
case False
from mem-eq subset $\mathrm{Read}_{\mathsf{sb}}$ False non-vol **have** meq: m′ a = m a
by (clarsimp)
from subset **obtain**
subset′: unforwarded-non-volatile-reads sb W $\subseteq$ A
using non-vol False
by (auto simp add: $\mathrm{Read}_{\mathsf{sb}}$ non-vol split: if-split-asm)
from mem-eq non-vol **have** mem-eq′: $\forall$ a′ $\in$ A $\cup$ W. m′ a′ = m a′
by auto
with Cons.hyps [OF mem-eq′ subset′ consis′] v meq
show ?thesis
by (simp add: $\mathrm{Read}_{\mathsf{sb}}$ non-vol False)
qed

qed
next
case $Prog_{sb}$ **with** Cons **show** ?thesis **by** auto
next
case $Ghost_{sb}$ **with** Cons **show** ?thesis **by** auto
qed
qed

lemma reads-consistent-mem-eq-on-unforwarded-non-volatile-reads-drop:
assumes mem-eq: $\forall a \in A \cup W.\ m' a = m a$
assumes subset: unforwarded-non-volatile-reads (dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb) W $\subseteq$ A
assumes subset-acq: acquired-reads True (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb) $\mathcal{O}$ $\subseteq$ A
assumes consis-m: reads-consistent False $\mathcal{O}$ m sb
shows reads-consistent False $\mathcal{O}$ m′ sb
using mem-eq subset subset-acq consis-m
proof (induct sb arbitrary: A W m′ m $\mathcal{O}$)
case Nil **thus** ?case **by** simp
next
case (Cons r sb)
note mem-eq = ‹$\forall a \in A \cup W.\ m' a = m a$›
note subset = ‹unforwarded-non-volatile-reads
(dropWhile (Not ∘ is-volatile-$Write_{sb}$) (r#sb)) W $\subseteq$ A›
note subset-acq = ‹acquired-reads True (takeWhile (Not ∘ is-volatile-$Write_{sb}$)(r#sb)) $\mathcal{O}$ $\subseteq$ A›
note consis-m = ‹reads-consistent False $\mathcal{O}$ m (r#sb)›

show ?case
proof (cases r)
case ($Write_{sb}$ volatile a sop v A′ L R W′)
show ?thesis
proof (cases volatile)
case True
from mem-eq
have mem-eq′:
$\forall a' \in (A \cup (\text{insert a W})).\ (m'(a:=v))\ a' = (m(a:=v))\ a'$
by (auto)

from consis-m **obtain**
consis′: reads-consistent True ($\mathcal{O} \cup A' - R$) (m(a := v)) sb **and**
no-volatile-$Read_{sb}$: outstanding-refs is-volatile-$Read_{sb}$ sb = {}
by (simp add: $Write_{sb}$ True)

from subset **obtain** unforwarded-non-volatile-reads sb (insert a W) $\subseteq$ A
by (clarsimp simp add: $Write_{sb}$ True)

from reads-consistent-mem-eq-on-unforwarded-non-volatile-reads [OF mem-eq′ this consis′]

have reads-consistent True ($\mathcal{O} \cup A' - R$) ($m'(a := v)$) sb.
with no-volatile-Read$_{sb}$
show ?thesis
by (simp add: Write$_{sb}$ True)
next
case False
from mem-eq
have mem-eq′:
$\forall a' \in (A \cup W).\ (m'(a{:=}v))\ a' = (m(a{:=}v))\ a'$
by (auto)
from subset **obtain**
subset′: unforwarded-non-volatile-reads (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb) W $\subseteq$ A
by (auto simp add: Write$_{sb}$ False)
from subset-acq **have**
subset-acq′: acquired-reads True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{O} \subseteq$ A
by (auto simp add: Write$_{sb}$ False)

from consis-m **obtain** consis′: reads-consistent False $\mathcal{O}$ ($m(a := v)$) sb
by (simp add: Write$_{sb}$ False)
from Cons.hyps [OF mem-eq′ subset′ subset-acq′ consis′]
have reads-consistent False $\mathcal{O}$ ($m'(a := v)$) sb.
then
show ?thesis
by (simp add: Write$_{sb}$ False)
qed
next
case (Read$_{sb}$ volatile a t v)
from mem-eq
have mem-eq′:
$\forall a' \in A \cup W.\ m'\ a' = m\ a'$
by (auto)
from subset **obtain**
subset′: unforwarded-non-volatile-reads (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb) W $\subseteq$ A
by (clarsimp simp add: Read$_{sb}$)
from subset-acq **obtain**
a-A: $\neg$ volatile $\longrightarrow a \in \mathcal{O} \longrightarrow a \in A$ **and**
subset-acq′: acquired-reads True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{O} \subseteq$ A
by (auto simp add: Read$_{sb}$ split: if-split-asm)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from consis-m **obtain**
consis′: reads-consistent False $\mathcal{O}$ m sb
by (simp add: Read$_{sb}$ True)

from Cons.hyps [OF mem-eq′ subset′ subset-acq′ consis′]
show ?thesis

by (simp add: $Read_{sb}$ True)
next
case False
note non-vol = this
from consis-m **obtain**
consis′: reads-consistent False $\mathcal{O}$ m sb **and**
v: a ∈ $\mathcal{O}$ ⟶ v=m a
by (simp add: $Read_{sb}$ False)

from mem-eq a-A v **have** v′: a ∈ $\mathcal{O}$ ⟶ v=m′ a
by (auto simp add: non-vol)
from Cons.hyps [OF mem-eq′ subset′ subset-acq′ consis′] v′
show ?thesis
by (simp add: $Read_{sb}$ False)
qed
next
case $Prog_{sb}$ **with** Cons **show** ?thesis **by** auto
next
case $Ghost_{sb}$ **with** Cons **show** ?thesis **by** auto
qed
qed

lemma read-only-read-witness:⋀$\mathcal{S}$ $\mathcal{O}$.
⟦non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb;
a ∈ read-only-reads $\mathcal{O}$ sb⟧
⟹
∃xs ys t v. sb=xs@ $Read_{sb}$ False a t v # ys ∧ a ∈ read-only (share xs $\mathcal{S}$) ∧ a ∉ read-only-reads $\mathcal{O}$ xs
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case ($Write_{sb}$ volatile a′ sop v A L R W)
show ?thesis
proof (cases volatile)
case True

from Cons.prems **obtain**
a-ro: a ∈ read-only-reads ($\mathcal{O}$ ∪ A − R) sb **and**
nvo′: non-volatile-owned-or-read-only True ($\mathcal{S}$ $\oplus_{W}$ R $\ominus_{A}$ L) ($\mathcal{O}$ ∪ A − R) sb
by (clarsimp simp add: $Write_{sb}$ True)

from Cons.hyps [OF nvo′ a-ro]

obtain xs ys t v **where**
sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t v # ys ∧ a ∈ read-only (share xs ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) ∧
a ∉ read-only-reads ($\mathcal{O}$ ∪ A − R) xs
by blast

thus ?thesis
apply −
apply (rule-tac x=(x#xs) **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v **in** exI)
apply (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ True)
done
next
case False
from Cons.prems **obtain**
a-ro: a ∈ read-only-reads $\mathcal{O}$ sb **and**
nvo′: non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ False)

from Cons.hyps [OF nvo′ a-ro]
obtain xs ys t v **where**
sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t v # ys ∧ a ∈ read-only (share xs $\mathcal{S}$) ∧ a ∉ read-only-reads $\mathcal{O}$
xs
by blast

thus ?thesis
apply −
apply (rule-tac x=(x#xs) **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v **in** exI)
apply (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ False)
done
qed
next
case ($\text{Read}_{\mathsf{sb}}$ volatile a′ t v)
show ?thesis
proof (cases a′=a ∧ a ∉ $\mathcal{O}$ ∧ ¬ volatile)
case True
with Cons.prems **have** a ∈ read-only $\mathcal{S}$
by (simp add: $\text{Read}_{\mathsf{sb}}$)

with True **show** ?thesis
apply −
apply (rule-tac x=[] **in** exI)
apply (rule-tac x=sb **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v **in** exI)
apply (clarsimp simp add: $\text{Read}_{\mathsf{sb}}$)

done
next
case False
with Cons.prems **obtain**
a-ro: $a \in$ read-only-reads $\mathcal{O}$ sb **and**
nvo$'$: non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb
by (auto simp add: $\text{Read}_{\mathsf{sb}}$ split: if-split-asm)
from Cons.hyps [OF nvo$'$ a-ro]
obtain xs ys t$'$ v$'$ **where**
sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t$'$ v$'$ # ys $\wedge$ a $\in$ read-only (share xs $\mathcal{S}$) $\wedge$ a $\notin$ read-only-reads $\mathcal{O}$ xs
by blast

with False **show** ?thesis
apply –
apply (rule-tac x=(x#xs) **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t$'$ **in** exI)
apply (rule-tac x=v$'$ **in** exI)
apply (clarsimp simp add: $\text{Read}_{\mathsf{sb}}$)
done
qed
next
case $\text{Prog}_{\mathsf{sb}}$
from Cons.prems **obtain**
a-ro: $a \in$ read-only-reads $\mathcal{O}$ sb **and**
nvo$'$: non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb
by (clarsimp simp add: $\text{Prog}_{\mathsf{sb}}$)

from Cons.hyps [OF nvo$'$ a-ro]
obtain xs ys t v **where**
sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t v # ys $\wedge$ a $\in$ read-only (share xs $\mathcal{S}$) $\wedge$ a $\notin$ read-only-reads $\mathcal{O}$ xs
by blast

thus ?thesis
apply –
apply (rule-tac x=(x#xs) **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v **in** exI)
apply (clarsimp simp add: $\text{Prog}_{\mathsf{sb}}$)
done
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.prems **obtain**
a-ro: $a \in$ read-only-reads ($\mathcal{O} \cup$ A $-$ R) sb **and**
nvo$'$: non-volatile-owned-or-read-only True ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)

from Cons.hyps [OF nvo′ a-ro]
obtain xs ys t v **where**
sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t v # ys ∧ a ∈ read-only (share xs ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) ∧ a ∉ read-only-reads ($\mathcal{O}$ ∪ A − R) xs
by blast

thus ?thesis
apply −
apply (rule-tac x=(x#xs) **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v **in** exI)
apply (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
done
qed
qed

lemma read-only-read-acquired-witness: ⋀$\mathcal{S}$ $\mathcal{O}$.
⟦non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb; sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb;
a ∉ read-only $\mathcal{S}$; a ∉ $\mathcal{O}$; a ∈ read-only-reads $\mathcal{O}$ sb⟧
⟹
∃xs ys t v. sb=xs@ $\text{Read}_{\mathsf{sb}}$ False a t v # ys ∧ a ∈ all-acquired xs ∧ a ∈ read-only (share xs $\mathcal{S}$) ∧
a ∉ read-only-reads $\mathcal{O}$ xs
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a′ sop v A L R W)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems **obtain**
nvo′: non-volatile-owned-or-read-only True ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ ∪ A − R) sb **and**
a-nro: a ∉ read-only $\mathcal{S}$ **and**
a-unowned: a ∉ $\mathcal{O}$ **and**
a-ro′: a ∈ read-only-reads ($\mathcal{O}$ ∪ A − R) sb **and**
A-shared-owns: A ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$ **and** L-A: L ⊆ A **and** A-R: A ∩ R = {} **and**
R-owns: R ⊆ $\mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ ∪ A − R) sb
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ True)

from R-owns a-unowned
have a-R: a ∉ R
by auto
show ?thesis

proof (cases a ∈ A)
case True
from read-only-read-witness [OF nvo′ a-ro′]
obtain xs ys t v′ **where**
sb: sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t v′ # ys **and**
ro: a ∈ read-only (share xs ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) **and**
a-ro-xs: a ∉ read-only-reads ($\mathcal{O}$ ∪ A − R) xs
by blast

with True **show** ?thesis
apply −
apply (rule-tac x=x#xs **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v′ **in** exI)
apply (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ volatile)
done
next
case False
with a-unowned R-owns a-nro L-A A-R
obtain a-nro′: a ∉ read-only ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) **and** a-unowned′: a ∉ $\mathcal{O}$ ∪ A − R
by (force simp add: in-read-only-convs)

from Cons.hyps [OF nvo′ consis′ a-nro′ a-unowned′ a-ro′]
obtain xs ys t v′ **where** sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t v′ # ys ∧
a ∈ all-acquired xs ∧ a ∈ read-only (share xs ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) ∧
a ∉ read-only-reads ($\mathcal{O}$ ∪ A − R) xs
by blast

then show ?thesis
apply −
apply (rule-tac x=x#xs **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v′ **in** exI)
apply (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ volatile)
done
qed
next
case False
from Cons.prems **obtain**
consis′: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb **and**
a-nro′: a ∉ read-only $\mathcal{S}$ **and**
a-unowned: a ∉ $\mathcal{O}$ **and**
a-ro′: a ∈ read-only-reads $\mathcal{O}$ sb **and**
a′ ∈ $\mathcal{O}$ **and**
nvo′: non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ False)

from Cons.hyps [OF nvo′ consis′ a-nro′ a-unowned a-ro′]

obtain xs ys t v′ **where**
sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t v′ # ys ∧
a ∈ all-acquired xs ∧ a ∈ read-only (share xs $\mathcal{S}$) ∧ a ∉ read-only-reads $\mathcal{O}$ xs
by blast

then show ?thesis
apply −
apply (rule-tac x=x#xs **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v′ **in** exI)
apply (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ False)
done
qed
next
case ($\text{Read}_{\mathsf{sb}}$ volatile a′ t v)
from Cons.prems
obtain
consis′: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb **and**
a-nro′: a ∉ read-only $\mathcal{S}$ **and**
a-unowned: a ∉ $\mathcal{O}$ **and**
a-ro′: a ∈ read-only-reads $\mathcal{O}$ sb **and**
nvo′: non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb
by (auto simp add: $\text{Read}_{\mathsf{sb}}$ split: if-split-asm)

from Cons.hyps [OF nvo′ consis′ a-nro′ a-unowned a-ro′]
obtain xs ys t v′ **where**
sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t v′ # ys ∧
a ∈ all-acquired xs ∧ a ∈ read-only (share xs $\mathcal{S}$) ∧ a ∉ read-only-reads $\mathcal{O}$ xs
by blast

with Cons.prems **show** ?thesis
apply −
apply (rule-tac x=x#xs **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v′ **in** exI)
apply (clarsimp simp add: $\text{Read}_{\mathsf{sb}}$)
done
next
case $\text{Prog}_{\mathsf{sb}}$
from Cons.prems
obtain
consis′: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb **and**
a-nro′: a ∉ read-only $\mathcal{S}$ **and**
a-unowned: a ∉ $\mathcal{O}$ **and**
a-ro′: a ∈ read-only-reads $\mathcal{O}$ sb **and**
nvo′: non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb
by (auto simp add: $\text{Prog}_{\mathsf{sb}}$)

from Cons.hyps [OF nvo′ consis′ a-nro′ a-unowned a-ro′]
obtain xs ys t v **where**
sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t v # ys ∧
a ∈ all-acquired xs ∧ a ∈ read-only (share xs $\mathcal{S}$) ∧ a ∉ read-only-reads $\mathcal{O}$ xs
by blast

then show ?thesis
apply −
apply (rule-tac x=x#xs **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v **in** exI)
apply (clarsimp simp add: $\text{Prog}_{\mathsf{sb}}$)
done
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.prems **obtain**
nvo′: non-volatile-owned-or-read-only True ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ ∪ A − R) sb **and**
a-nro: a ∉ read-only $\mathcal{S}$ **and**
a-unowned: a ∉ $\mathcal{O}$ **and**
a-ro′: a ∈ read-only-reads ($\mathcal{O}$ ∪ A − R) sb **and**
A-shared-owns: A ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$ **and** L-A: L ⊆ A **and** A-R: A ∩ R = {} **and**
R-owns: R ⊆ $\mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}$ ∪ A − R) sb
by (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)

from R-owns a-unowned
have a-R: a ∉ R
by auto
show ?thesis
proof (cases a ∈ A)
case True
from read-only-read-witness [OF nvo′ a-ro′]
obtain xs ys t v′ **where**
sb: sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t v′ # ys **and**
ro: a ∈ read-only (share xs ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) **and**
a-ro-xs: a ∉ read-only-reads ($\mathcal{O}$ ∪ A − R) xs
by blast

with True **show** ?thesis
apply −
apply (rule-tac x=x#xs **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v′ **in** exI)
apply (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
done
next
case False
with a-unowned R-owns a-nro L-A A-R

obtain a-nro′: a ∉ read-only ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) **and** a-unowned′: a ∉ $\mathcal{O}$ ∪ A − R
by (force simp add: in-read-only-convs)

from Cons.hyps [OF nvo′ consis′ a-nro′ a-unowned′ a-ro′]
obtain xs ys t v′ **where** sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t v′ # ys ∧
a ∈ all-acquired xs ∧ a ∈ read-only (share xs ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) ∧
a ∉ read-only-reads ($\mathcal{O}$ ∪ A − R) xs
by blast

then show ?thesis
apply −
apply (rule-tac x=x#xs **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v′ **in** exI)
apply (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
done
qed
qed
qed

lemma unforwarded-not-written: ⋀W. a ∈ unforwarded-non-volatile-reads sb W ⟹ a ∉ W
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a′ sop v A L R W′)
from Cons.prems
have a ∈ unforwarded-non-volatile-reads sb (insert a′ W)
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$)
from Cons.hyps [OF this]
have a ∉ insert a′ W**.**
then show ?thesis
by blast
next
case ($\text{Read}_{\mathsf{sb}}$ volatile a′ t v)
with Cons.hyps [of W] Cons.prems **show** ?thesis
by (auto split: if-split-asm)
next
case $\text{Prog}_{\mathsf{sb}}$
with Cons.hyps [of W] Cons.prems **show** ?thesis
by (auto split: if-split-asm)
next
case $\text{Ghost}_{\mathsf{sb}}$
with Cons.hyps [of W] Cons.prems **show** ?thesis

by (auto split: if-split-asm)
qed
qed

lemma unforwarded-witness: $\bigwedge$X.
⟦a ∈ unforwarded-non-volatile-reads sb X⟧
$\Longrightarrow$
∃xs ys t v. sb=xs@ $\text{Read}_{\mathsf{sb}}$ False a t v # ys ∧ a ∉ outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ xs
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a′ sop v A L R W)
show ?thesis
proof (cases volatile)
case True

from Cons.prems **obtain**
a-unforw: a ∈ unforwarded-non-volatile-reads sb (insert a′ X)
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ True)

from unforwarded-not-written [OF a-unforw]
have a′-a: a′≠a
by auto

from Cons.hyps [OF a-unforw]
obtain xs ys t v **where**
sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t v # ys ∧
a ∉ outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ xs
by blast

thus ?thesis
using a′-a
apply –
apply (rule-tac x=(x#xs) **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v **in** exI)
apply (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ True)
done
next
case False
from Cons.prems **obtain**
a-unforw: a ∈ unforwarded-non-volatile-reads sb (insert a′ X)
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ False)

from unforwarded-not-written [OF a-unforw]

have a′-a: a′≠a
by auto

from Cons.hyps [OF a-unforw]
obtain xs ys t v **where**
sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t v # ys ∧
a ∉ outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ xs
by blast

thus ?thesis
using a′-a
apply –
apply (rule-tac x=(x#xs) **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v **in** exI)
apply (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ False)
done
qed
next
case ($\text{Read}_{\mathsf{sb}}$ volatile a′ t v)
show ?thesis
proof (cases a′=a ∧ a ∉ X ∧ ¬ volatile)
case True

with True **show** ?thesis
apply –
apply (rule-tac x=[] **in** exI)
apply (rule-tac x=sb **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v **in** exI)
apply (clarsimp simp add: $\text{Read}_{\mathsf{sb}}$)
done
next
case False
note not-ror = this
with Cons.prems **obtain** a-unforw: a ∈ unforwarded-non-volatile-reads sb X
by (auto simp add: $\text{Read}_{\mathsf{sb}}$ split: if-split-asm)

from Cons.hyps [OF a-unforw]
obtain xs ys t v **where**
sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t v # ys ∧
a ∉ outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ xs
by blast

thus ?thesis
apply –
apply (rule-tac x=(x#xs) **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)

```
 apply (rule-tac x=v in exI)
 apply (clarsimp simp add: Read_sb)
 done
   qed
  next
   case Prog_sb
   from Cons.prems obtain a-unforw: a ∈ unforwarded-non-volatile-reads sb X
     by (auto simp add: Prog_sb)

   from Cons.hyps [OF a-unforw]
   obtain xs ys t v where
     sb = xs @ Read_sb False a t v # ys ∧
     a ∉ outstanding-refs is-Write_sb xs
     by blast

   thus ?thesis
     apply −
     apply (rule-tac x=(x#xs) in exI)
     apply (rule-tac x=ys in exI)
     apply (rule-tac x=t in exI)
     apply (rule-tac x=v in exI)
     apply (clarsimp simp add: Prog_sb)
     done
  next
   case (Ghost_sb A L R W)
   from Cons.prems obtain a-unforw: a ∈ unforwarded-non-volatile-reads sb X
     by (auto simp add: Ghost_sb)

   from Cons.hyps [OF a-unforw]
   obtain xs ys t v where
     sb = xs @ Read_sb False a t v # ys ∧
     a ∉ outstanding-refs is-Write_sb xs
     by blast

   thus ?thesis
     apply −
     apply (rule-tac x=(x#xs) in exI)
     apply (rule-tac x=ys in exI)
     apply (rule-tac x=t in exI)
     apply (rule-tac x=v in exI)
     apply (clarsimp simp add: Ghost_sb)
     done
  qed
qed


lemma read-only-read-acquired-unforwarded-witness: ⋀𝒮 𝒪 X.
  ⟦non-volatile-owned-or-read-only True 𝒮 𝒪 sb; sharing-consistent 𝒮 𝒪 sb;
  a ∉ read-only 𝒮; a ∉ 𝒪; a ∈ read-only-reads 𝒪 sb;
  a ∈ unforwarded-non-volatile-reads sb X ⟧
```

$\Longrightarrow$

$\exists$xs ys t v. sb=xs@ Read_{sb} False a t v # ys $\wedge$ a $\in$ all-acquired xs $\wedge$

a $\notin$ outstanding-refs is-Write_{sb} xs

proof (induct sb)

case Nil **thus** ?case **by** simp

next

case (Cons x sb)

show ?case

proof (cases x)

case (Write_{sb} volatile a$'$ sop v A L R W)

show ?thesis

proof (cases volatile)

case True

note volatile=this

from Cons.prems **obtain**

nvo$'$: non-volatile-owned-or-read-only True ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb **and**

a-nro: a $\notin$ read-only $\mathcal{S}$ **and**

a-unowned: a $\notin \mathcal{O}$ **and**

a-ro$'$: a $\in$ read-only-reads ($\mathcal{O} \cup$ A $-$ R) sb **and**

A-shared-owns: A $\subseteq$ dom $\mathcal{S} \cup \mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and**

R-owns: R $\subseteq \mathcal{O}$ **and**

consis$'$: sharing-consistent ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb **and**

a-unforw: a $\in$ unforwarded-non-volatile-reads sb (insert a$'$ X)

by (clarsimp simp add: Write_{sb} True)

from unforwarded-not-written [OF a-unforw]

have a-notin: a $\notin$ insert a$'$ X**.**

from R-owns a-unowned

have a-R: a $\notin$ R

by auto

show ?thesis

proof (cases a $\in$ A)

case True

from unforwarded-witness [OF a-unforw]

obtain xs ys t v$'$ **where**

sb: sb = xs @ Read_{sb} False a t v$'$ # ys **and**

a-xs: a $\notin$ outstanding-refs is-Write_{sb} xs

by blast

with True a-notin **show** ?thesis

apply $-$

apply (rule-tac x=x#xs **in** exI)

apply (rule-tac x=ys **in** exI)

apply (rule-tac x=t **in** exI)

apply (rule-tac x=v$'$ **in** exI)

apply (clarsimp simp add: Write_{sb} volatile)

done

next

case False

with a-unowned R-owns a-nro L-A A-R
obtain a-nro′: $a \notin$ read-only ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) **and** a-unowned′: $a \notin \mathcal{O} \cup A - R$
by (force simp add: in-read-only-convs)

from Cons.hyps [OF nvo′ consis′ a-nro′ a-unowned′ a-ro′ a-unforw]
obtain xs ys t v′ **where** sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t v′ # ys $\wedge$
$a \in$ all-acquired xs $\wedge$
$a \notin$ outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ xs
by blast

with a-notin **show** ?thesis
apply −
apply (rule-tac x=x#xs **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v′ **in** exI)
apply (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ volatile)
done
qed
next
case False
from Cons.prems **obtain**
consis′: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb **and**
a-nro′: $a \notin$ read-only $\mathcal{S}$ **and**
a-unowned: $a \notin \mathcal{O}$ **and**
a-ro′: $a \in$ read-only-reads $\mathcal{O}$ sb **and**
$a' \in \mathcal{O}$ **and**
nvo′: non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb **and**
a-unforw′: $a \in$ unforwarded-non-volatile-reads sb (insert a′ X)
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ False split: if-split-asm)

from unforwarded-not-written [OF a-unforw′]
have a-notin: $a \notin$ insert a′ X**.**

from Cons.hyps [OF nvo′ consis′ a-nro′ a-unowned a-ro′ a-unforw′]
obtain xs ys t v′ **where**
sb = xs @ $\text{Read}_{\mathsf{sb}}$ False a t v′ # ys $\wedge$
$a \in$ all-acquired xs $\wedge$ $a \notin$ outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ xs
by blast

with a-notin **show** ?thesis
apply −
apply (rule-tac x=x#xs **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v′ **in** exI)
apply (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ False)
done
qed
next

case ($Read_{sb}$ volatile a′ t v)
from Cons.prems
obtain
consis′: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb **and**
a-nro′: a $\notin$ read-only $\mathcal{S}$ **and**
a-unowned: a $\notin$ $\mathcal{O}$ **and**
a-ro′: a $\in$ read-only-reads $\mathcal{O}$ sb **and**
nvo′: non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb **and**
a-unforw: a $\in$ unforwarded-non-volatile-reads sb X
by (auto simp add: $Read_{sb}$ split: if-split-asm)

from Cons.hyps [OF nvo′ consis′ a-nro′ a-unowned a-ro′ a-unforw]
obtain xs ys t v′ **where**
sb = xs @ $Read_{sb}$ False a t v′ # ys $\wedge$
a $\in$ all-acquired xs $\wedge$ a $\notin$ outstanding-refs is-$Write_{sb}$ xs
by blast

with Cons.prems **show** ?thesis
apply −
apply (rule-tac x=x#xs **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v′ **in** exI)
apply (clarsimp simp add: $Read_{sb}$)
done
next
case $Prog_{sb}$
from Cons.prems
obtain
consis′: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb **and**
a-nro′: a $\notin$ read-only $\mathcal{S}$ **and**
a-unowned: a $\notin$ $\mathcal{O}$ **and**
a-ro′: a $\in$ read-only-reads $\mathcal{O}$ sb **and**
nvo′: non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb **and**
a-unforw: a $\in$ unforwarded-non-volatile-reads sb X
by (auto simp add: $Prog_{sb}$)

from Cons.hyps [OF nvo′ consis′ a-nro′ a-unowned a-ro′ a-unforw]
obtain xs ys t v **where**
sb = xs @ $Read_{sb}$ False a t v # ys $\wedge$
a $\in$ all-acquired xs $\wedge$ a $\notin$ outstanding-refs is-$Write_{sb}$ xs
by blast

then show ?thesis
apply −
apply (rule-tac x=x#xs **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v **in** exI)
apply (clarsimp simp add: $Prog_{sb}$)

done
next
case ($Ghost_{sb}$ A L R W)
from Cons.prems **obtain**
nvo′: non-volatile-owned-or-read-only True ($\mathcal{S} \oplus_{W}$ R $\ominus_{A}$ L) ($\mathcal{O}$ ∪ A − R) sb **and**
a-nro: a ∉ read-only $\mathcal{S}$ **and**
a-unowned: a ∉ $\mathcal{O}$ **and**
a-ro′: a ∈ read-only-reads ($\mathcal{O}$ ∪ A − R) sb **and**
A-shared-owns: A ⊆ dom $\mathcal{S}$ ∪ $\mathcal{O}$ **and** L-A: L ⊆ A **and** A-R: A ∩ R = {} **and**
R-owns: R ⊆ $\mathcal{O}$ **and**
consis′: sharing-consistent ($\mathcal{S} \oplus_{W}$ R $\ominus_{A}$ L) ($\mathcal{O}$ ∪ A − R) sb **and**
a-unforw: a ∈ unforwarded-non-volatile-reads sb (X)
by (clarsimp simp add: $Ghost_{sb}$)

from unforwarded-not-written [OF a-unforw]
have a-notin: a ∉ X**.**
from R-owns a-unowned
have a-R: a ∉ R
by auto
show ?thesis
proof (cases a ∈ A)
case True

from unforwarded-witness [OF a-unforw]
obtain xs ys t v′ **where**
sb: sb = xs @ $Read_{sb}$ False a t v′ # ys **and**
a-xs: a ∉ outstanding-refs is-$Write_{sb}$ xs
by blast

with True a-notin **show** ?thesis
apply −
apply (rule-tac x=x#xs **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v′ **in** exI)
apply (clarsimp simp add: $Ghost_{sb}$)
done
next
case False
with a-unowned R-owns a-nro L-A A-R
obtain a-nro′: a ∉ read-only ($\mathcal{S} \oplus_{W}$ R $\ominus_{A}$ L) **and** a-unowned′: a ∉ $\mathcal{O}$ ∪ A − R
by (force simp add: in-read-only-convs)

from Cons.hyps [OF nvo′ consis′ a-nro′ a-unowned′ a-ro′ a-unforw]
obtain xs ys t v′ **where** sb = xs @ $Read_{sb}$ False a t v′ # ys ∧
a ∈ all-acquired xs ∧
a ∉ outstanding-refs is-$Write_{sb}$ xs
by blast

with a-notin **show** ?thesis

apply −
apply (rule-tac x=x#xs **in** exI)
apply (rule-tac x=ys **in** exI)
apply (rule-tac x=t **in** exI)
apply (rule-tac x=v′ **in** exI)
apply (clarsimp simp add: $\text{Ghost}_{\mathsf{sb}}$)
done
qed
qed
qed

lemma takeWhile-prefix: ∃ys. takeWhile P xs @ ys = xs
apply (induct xs)
apply auto
done

lemma unforwarded-empty-extend:
⋀W. x ∈ unforwarded-non-volatile-reads sb {} ⟹ x ∉ W ⟹ x ∈ unforwarded-non-volatile-reads sb W
apply (induct sb)
apply clarsimp
subgoal for a sb W
apply (case-tac a)
apply clarsimp
apply (frule unforwarded-not-written)
apply (drule-tac W={} **in** unforwarded-non-volatile-reads-antimono-in)
apply blast
apply (auto split: if-split-asm)
done
done

lemma notin-unforwarded-empty:
⋀W. a ∉ unforwarded-non-volatile-reads sb W ⟹ a ∉ W ⟹ a ∉ unforwarded-non-volatile-reads sb {}
using unforwarded-empty-extend
by blast

lemma
assumes ro: a ∈ read-only $\mathcal{S}$ ⟶ a ∈ read-only $\mathcal{S}'$
assumes a-in: a ∈ read-only ($\mathcal{S} \oplus_{\mathsf{W}}$ R)
shows a ∈ read-only ($\mathcal{S}' \oplus_{\mathsf{W}}$ R)
using ro a-in
by (auto simp add: in-read-only-convs)

lemma
assumes ro: a ∈ read-only $\mathcal{S}$ ⟶ a ∈ read-only $\mathcal{S}'$
assumes a-in: a ∈ read-only ($\mathcal{S} \ominus_{\mathsf{A}}$ L)
shows a ∈ read-only ($\mathcal{S}' \ominus_{\mathsf{A}}$ L)
using ro a-in

by (auto simp add: in-read-only-convs)

lemma non-volatile-owned-or-read-only-read-only-reads-eq:
$\bigwedge\mathcal{S}\ \mathcal{S}'\ \mathcal{O}$ pending-write.
$[\![$non-volatile-owned-or-read-only pending-write $\mathcal{S}\ \mathcal{O}$ sb;
$\forall$a $\in$ read-only-reads $\mathcal{O}$ sb. a $\in$ read-only $\mathcal{S} \longrightarrow$ a $\in$ read-only $\mathcal{S}'$
$]\!]$
$\Longrightarrow$ non-volatile-owned-or-read-only pending-write $\mathcal{S}'\ \mathcal{O}$ sb
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case (Write$_{\mathsf{sb}}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems **obtain**
nvo′: non-volatile-owned-or-read-only True ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O} \cup$ A $-$ R) sb **and**
ro′: $\forall$a$\in$read-only-reads ($\mathcal{O} \cup$ A $-$ R) sb. a $\in$ read-only $\mathcal{S} \longrightarrow$ a $\in$ read-only $\mathcal{S}'$
by (clarsimp simp add: Write$_{\mathsf{sb}}$ volatile)

from ro′
have ro′′:$\forall$a$\in$read-only-reads ($\mathcal{O} \cup$ A $-$ R) sb.
a $\in$ read-only ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $\longrightarrow$ a $\in$ read-only ($\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (auto simp add: in-read-only-convs)
from Cons.hyps [OF nvo′ ro′′]
show ?thesis
by (clarsimp simp add: Write$_{\mathsf{sb}}$ volatile)
next
case False
with Cons.hyps [of pending-write $\mathcal{S}\ \mathcal{O}\ \mathcal{S}'$] Cons.prems **show** ?thesis
by (auto simp add: Write$_{\mathsf{sb}}$)
qed
next
case (Read$_{\mathsf{sb}}$ volatile a t v)
show ?thesis
proof (cases volatile)
case True
with Cons.hyps [of pending-write $\mathcal{S}\ \mathcal{O}\ \mathcal{S}'$] Cons.prems **show** ?thesis
by (auto simp add: Read$_{\mathsf{sb}}$)
next
case False
note non-vol = this
show ?thesis
proof (cases a $\in \mathcal{O}$)
case True
with Cons.hyps [of pending-write $\mathcal{S}\ \mathcal{O}\ \mathcal{S}'$] Cons.prems **show** ?thesis

by (auto simp add: $\text{Read}_{\mathsf{sb}}$ non-vol)
next
case False
from Cons.prems Cons.hyps [of pending-write $\mathcal{S}$ $\mathcal{O}$ $\mathcal{S}'$] **show** ?thesis
by (clarsimp simp add: $\text{Read}_{\mathsf{sb}}$ non-vol False)
qed
qed
next
case $\text{Prog}_{\mathsf{sb}}$
with Cons.hyps [of pending-write $\mathcal{S}$ $\mathcal{O}$ $\mathcal{S}'$] Cons.prems **show** ?thesis
by (auto)
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.hyps [of pending-write ($\mathcal{S} \oplus_{\mathsf{W}} \text{R} \ominus_{\mathsf{A}} \text{L}$) $\mathcal{O} \cup \text{A} - \text{R}$ $\mathcal{S}' \oplus_{\mathsf{W}} \text{R} \ominus_{\mathsf{A}} \text{L}$] Cons.prems
show ?thesis
by (auto simp add: $\text{Ghost}_{\mathsf{sb}}$ in-read-only-convs)
qed
qed

lemma non-volatile-owned-or-read-only-read-only-reads-eq$'$:
$\bigwedge \mathcal{S}\ \mathcal{S}'\ \mathcal{O}$.
⟦non-volatile-owned-or-read-only False $\mathcal{S}$ $\mathcal{O}$ sb;
$\forall$a $\in$ read-only-reads (acquired True (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) $\mathcal{O}$)
(dropWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb). a $\in$ read-only $\mathcal{S}$ $\longrightarrow$ a $\in$ read-only $\mathcal{S}'$
⟧
$\Longrightarrow$ non-volatile-owned-or-read-only False $\mathcal{S}'$ $\mathcal{O}$ sb
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems **obtain**
nvo$'$: non-volatile-owned-or-read-only True ($\mathcal{S} \oplus_{\mathsf{W}} \text{R} \ominus_{\mathsf{A}} \text{L}$) ($\mathcal{O} \cup \text{A} - \text{R}$) sb **and**
ro$'$: $\forall$a$\in$read-only-reads ($\mathcal{O} \cup \text{A} - \text{R}$) sb. a $\in$ read-only $\mathcal{S}$ $\longrightarrow$ a $\in$ read-only $\mathcal{S}'$
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ volatile)

from ro$'$
have ro$''$:$\forall$a$\in$read-only-reads ($\mathcal{O} \cup \text{A} - \text{R}$) sb.
a $\in$ read-only ($\mathcal{S} \oplus_{\mathsf{W}} \text{R} \ominus_{\mathsf{A}} \text{L}$) $\longrightarrow$ a $\in$ read-only ($\mathcal{S}' \oplus_{\mathsf{W}} \text{R} \ominus_{\mathsf{A}} \text{L}$)
by (auto simp add: in-read-only-convs)
from non-volatile-owned-or-read-only-read-only-reads-eq [OF nvo$'$ ro$''$]
show ?thesis

by (clarsimp simp add: $\mathsf{Write}_{\mathsf{sb}}$ volatile)
next
case False
with Cons.hyps [of $\mathcal{S}$ $\mathcal{O}$ $\mathcal{S}'$] Cons.prems **show** ?thesis
by (auto simp add: $\mathsf{Write}_{\mathsf{sb}}$)
qed
next
case ($\mathsf{Read}_{\mathsf{sb}}$ volatile a t v)
show ?thesis
proof (cases volatile)
case True
with Cons.hyps [of $\mathcal{S}$ $\mathcal{O}$ $\mathcal{S}'$] Cons.prems **show** ?thesis
by (auto simp add: $\mathsf{Read}_{\mathsf{sb}}$)
next
case False
note non-vol = this
show ?thesis
proof (cases a $\in$ $\mathcal{O}$)
case True
with Cons.hyps [of $\mathcal{S}$ $\mathcal{O}$ $\mathcal{S}'$] Cons.prems **show** ?thesis
by (auto simp add: $\mathsf{Read}_{\mathsf{sb}}$ non-vol)
next
case False
from Cons.prems Cons.hyps [of $\mathcal{S}$ $\mathcal{O}$ $\mathcal{S}'$] **show** ?thesis
by (clarsimp simp add: $\mathsf{Read}_{\mathsf{sb}}$ non-vol False)
qed
qed
next
case $\mathsf{Prog}_{\mathsf{sb}}$
with Cons.hyps [of $\mathcal{S}$ $\mathcal{O}$ $\mathcal{S}'$] Cons.prems **show** ?thesis
by (auto)
next
case ($\mathsf{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.hyps [of ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $\mathcal{O}$ $\cup$ A $-$ R $\mathcal{S}'$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L] Cons.prems
show ?thesis
by (auto simp add: $\mathsf{Ghost}_{\mathsf{sb}}$ in-read-only-convs)
qed
qed

lemma no-write-to-read-only-memory-read-only-reads-eq:
$\bigwedge\mathcal{S}$ $\mathcal{S}'$.
$[\![$no-write-to-read-only-memory $\mathcal{S}$ sb;
$\forall$a $\in$ outstanding-refs is-$\mathsf{Write}_{\mathsf{sb}}$ sb. a $\in$ read-only $\mathcal{S}'$ $\longrightarrow$ a $\in$ read-only $\mathcal{S}$
$]\!]$
$\Longrightarrow$ no-write-to-read-only-memory $\mathcal{S}'$ sb
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)

show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile a sop v A L R W)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems **obtain**
nvo′: no-write-to-read-only-memory ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) sb **and**
ro′: ∀a∈outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ sb. a ∈ read-only $\mathcal{S}'$ ⟶ a ∈ read-only $\mathcal{S}$ **and**
not-ro: a ∉ read-only $\mathcal{S}'$
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ volatile)

from ro′
have ro″:∀a∈outstanding-refs is-$\text{Write}_{\mathsf{sb}}$ sb.
a ∈ read-only ($\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ⟶ a ∈ read-only ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (auto simp add: in-read-only-convs)
from Cons.hyps [OF nvo′ ro″] not-ro
show ?thesis
by (clarsimp simp add: $\text{Write}_{\mathsf{sb}}$ volatile)
next
case False
with Cons.hyps [of $\mathcal{S}$ $\mathcal{S}'$] Cons.prems **show** ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$)
qed
next
case ($\text{Read}_{\mathsf{sb}}$ volatile a t v)
with Cons.hyps [of $\mathcal{S}$ $\mathcal{S}'$] Cons.prems **show** ?thesis
by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
next
case $\text{Prog}_{\mathsf{sb}}$
with Cons.hyps [of $\mathcal{S}$ $\mathcal{S}'$] Cons.prems **show** ?thesis
by (auto)
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
from Cons.hyps [of ($\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $\mathcal{S}' \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L] Cons.prems
show ?thesis
by (auto simp add: $\text{Ghost}_{\mathsf{sb}}$ in-read-only-convs)
qed
qed

lemma reads-consistent-drop:
reads-consistent False $\mathcal{O}$ m sb
⟹ reads-consistent True
(acquired True (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) $\mathcal{O}$)
(flush (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) m)
(dropWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb)
using reads-consistent-append [of False $\mathcal{O}$ m (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb)
(dropWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb)]

apply (cases outstanding-refs is-volatile-Write$_{sb}$ sb = {})
apply (clarsimp simp add: outstanding-vol-write-take-drop-appends
takeWhile-not-vol-write-outstanding-refs dropWhile-not-vol-write-empty)
apply(clarsimp simp add: outstanding-vol-write-take-drop-appends
takeWhile-not-vol-write-outstanding-refs dropWhile-not-vol-write-empty)
apply (case-tac (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb))
apply (fastforce simp add: outstanding-refs-conv)
apply (frule dropWhile-ConsD)
apply (clarsimp split: memref.splits)
done

lemma outstanding-refs-non-volatile-Read$_{sb}$-all-acquired-dropWhile′:
⋀m $\mathcal{S}$ $\mathcal{O}$ pending-write.
⟦reads-consistent pending-write $\mathcal{O}$ m sb;non-volatile-owned-or-read-only pending-write $\mathcal{S}$ $\mathcal{O}$ sb;
a ∈ outstanding-refs is-non-volatile-Read$_{sb}$ (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb)⟧
⟹ a ∈ $\mathcal{O}$ ∨ a ∈ all-acquired sb ∨
a ∈ read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{O}$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
proof (induct sb)
case Nil **thus** ?case **by** simp
next
case (Cons x sb)
show ?case
proof (cases x)
case (Write$_{sb}$ volatile a′ sop v A L R W)
show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.prems **obtain**
non-vo: non-volatile-owned-or-read-only True ($\mathcal{S}$ $\oplus_{W}$ R $\ominus_{A}$ L)
($\mathcal{O}$ ∪ A − R) sb **and**
out-vol: outstanding-refs is-volatile-Read$_{sb}$ sb = {} **and**
out: a ∈ outstanding-refs is-non-volatile-Read$_{sb}$ sb
by (clarsimp simp add: Write$_{sb}$ True)
show ?thesis
proof (cases a ∈ $\mathcal{O}$)
case True
show ?thesis
by (clarsimp simp add: Write$_{sb}$ True volatile)
next
case False
from outstanding-non-volatile-Read$_{sb}$-acquired-or-read-only-reads [OF non-vo out]
have a-in: a ∈ acquired-reads True sb ($\mathcal{O}$ ∪ A − R) ∨
a ∈ read-only-reads ($\mathcal{O}$ ∪ A − R) sb
by auto
with acquired-reads-all-acquired [of True sb ($\mathcal{O}$ ∪ A − R)]
show ?thesis

by (auto simp add: $\text{Write}_{\mathsf{sb}}$ volatile)
qed

next
case False
with Cons **show** ?thesis
by (auto simp add: $\text{Write}_{\mathsf{sb}}$ False)
qed
next
case $\text{Read}_{\mathsf{sb}}$
with Cons **show** ?thesis
apply (clarsimp simp del: o-apply simp add: $\text{Read}_{\mathsf{sb}}$
acquired-takeWhile-non-volatile-$\text{Write}_{\mathsf{sb}}$ split: if-split-asm)
apply auto
done
next
case $\text{Prog}_{\mathsf{sb}}$
with Cons **show** ?thesis
by (auto simp add: $\text{Read}_{\mathsf{sb}}$)
next
case ($\text{Ghost}_{\mathsf{sb}}$ A L R W)
with Cons.hyps [of pending-write $\mathcal{O} \cup A - R$ m $\mathcal{S} \oplus_{\mathsf{W}} R \ominus_{\mathsf{A}} L$]
read-only-reads-antimono [of $\mathcal{O}$ $\mathcal{O} \cup A - R$]
Cons.prems **show** ?thesis
by (auto simp add: $\text{Ghost}_{\mathsf{sb}}$)
qed
qed

end

theory ReduceStoreBufferSimulation
imports ReduceStoreBuffer
begin

locale $\text{initial}_{\mathsf{sb}}$ = simple-ownership-distinct + read-only-unowned + unowned-shared +
constrains ts::('p,'p store-buffer,bool,owns,rels) thread-config list
assumes empty-sb: $[\![$i < length ts; ts!i=(p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)$]\!] \Longrightarrow$ sb=[]
assumes empty-is: $[\![$i < length ts; ts!i=(p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)$]\!] \Longrightarrow$ is=[]
assumes empty-rels: $[\![$i < length ts; ts!i=(p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)$]\!] \Longrightarrow$ $\mathcal{R}$=Map.empty

sublocale $\text{initial}_{\mathsf{sb}} \subseteq$ outstanding-non-volatile-refs-owned-or-read-only
proof
fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p
assume i-bound: i < length ts
assume ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
show non-volatile-owned-or-read-only False $\mathcal{S}$ $\mathcal{O}$ sb
using empty-sb [OF i-bound ts-i] **by** auto
qed

sublocale initial$_{sb}$ $\subseteq$ outstanding-volatile-writes-unowned-by-others
proof
  fix i j p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j_i sb_i p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j
  assume i-bound: i < length ts **and**
    j-bound: j < length ts **and**
    neq-i-j: $i \neq j$ **and**
    ts-i: ts ! i = (p_i, is_i, j_i, sb_i, $\mathcal{D}_i$, $\mathcal{O}_i$, $\mathcal{R}_i$) **and**
    ts-j: ts ! j = (p_j, is_j, j_j, sb_j, $\mathcal{D}_j$, $\mathcal{O}_j$, $\mathcal{R}_j$)
  show ($\mathcal{O}_j$ $\cup$ all-acquired sb_j) $\cap$ outstanding-refs is-volatile-Write$_{sb}$ sb_i = {}
  using empty-sb [OF i-bound ts-i] empty-sb [OF j-bound ts-j] **by** auto
qed

sublocale initial$_{sb}$ $\subseteq$ read-only-reads-unowned
proof
  fix i j p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j_i sb_i p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j
  assume i-bound: i < length ts **and**
    j-bound: j < length ts **and**
    neq-i-j: $i \neq j$ **and**
    ts-i: ts ! i = (p_i, is_i, j_i, sb_i, $\mathcal{D}_i$, $\mathcal{O}_i$, $\mathcal{R}_i$) **and**
    ts-j: ts ! j = (p_j, is_j, j_j, sb_j, $\mathcal{D}_j$, $\mathcal{O}_j$, $\mathcal{R}_j$)
  show ($\mathcal{O}_j$ $\cup$ all-acquired sb_j) $\cap$
     read-only-reads (acquired True
                     (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_i) $\mathcal{O}_i$)
                     (dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_i) = {}
  using empty-sb [OF i-bound ts-i] empty-sb [OF j-bound ts-j] **by** auto
qed

sublocale initial$_{sb}$ $\subseteq$ ownership-distinct
proof
  fix i j p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j_i sb_i p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j
  assume i-bound: i < length ts **and**
    j-bound: j < length ts **and**
    neq-i-j: $i \neq j$ **and**
    ts-i: ts ! i = (p_i, is_i, j_i, sb_i, $\mathcal{D}_i$, $\mathcal{O}_i$, $\mathcal{R}_i$) **and**
    ts-j: ts ! j = (p_j, is_j, j_j, sb_j, $\mathcal{D}_j$, $\mathcal{O}_j$, $\mathcal{R}_j$)
  show ($\mathcal{O}_i$ $\cup$ all-acquired sb_i) $\cap$ ($\mathcal{O}_j$ $\cup$ all-acquired sb_j) = {}
   using simple-ownership-distinct [OF i-bound j-bound neq-i-j ts-i ts-j] empty-sb [OF i-bound ts-i] empty-sb
[OF j-bound ts-j]
  by auto
qed

sublocale initial$_{sb}$ $\subseteq$ valid-ownership **..**

sublocale initial$_{sb}$ $\subseteq$ outstanding-non-volatile-writes-unshared
proof
   fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p
   assume i-bound: i < length ts
   assume ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
   show non-volatile-writes-unshared $\mathcal{S}$ sb
   using empty-sb [OF i-bound ts-i] **by** auto
qed

sublocale initial$_{sb}$ $\subseteq$ sharing-consis
proof
   fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p
   assume i-bound: i < length ts
   assume ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
   show sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb
   using empty-sb [OF i-bound ts-i] **by** auto
qed

sublocale $\text{initial}_{\text{sb}}$ $\subseteq$ no-outstanding-write-to-read-only-memory
proof
  fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p
  assume i-bound: i $<$ length ts
  assume ts-i: ts!i $=$ (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  show no-write-to-read-only-memory $\mathcal{S}$ sb
  using empty-sb [OF i-bound ts-i] **by** auto
qed

sublocale $\text{initial}_{\text{sb}}$ $\subseteq$ valid-sharing ..
sublocale $\text{initial}_{\text{sb}}$ $\subseteq$ valid-ownership-and-sharing ..

sublocale $\text{initial}_{\text{sb}}$ $\subseteq$ load-tmps-distinct
proof
  fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p
  assume i-bound: i $<$ length ts
  assume ts-i: ts!i $=$ (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  show distinct-load-tmps is
  using empty-is [OF i-bound ts-i] **by** auto
qed

sublocale $\text{initial}_{\text{sb}}$ $\subseteq$ read-tmps-distinct
proof
  fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p
  assume i-bound: i $<$ length ts
  assume ts-i: ts!i $=$ (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  show distinct-read-tmps sb
  using empty-sb [OF i-bound ts-i] **by** auto
qed

sublocale $\text{initial}_{\text{sb}}$ $\subseteq$ load-tmps-read-tmps-distinct
proof
  fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p
  assume i-bound: i $<$ length ts
  assume ts-i: ts!i $=$ (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  show load-tmps is $\cap$ read-tmps sb $=$ {}
  using empty-sb [OF i-bound ts-i] empty-is [OF i-bound ts-i] **by** auto
qed

sublocale $\text{initial}_{\text{sb}}$ $\subseteq$ load-tmps-read-tmps-distinct ..

sublocale $\text{initial}_{\text{sb}}$ $\subseteq$ valid-write-sops
proof
  fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p
  assume i-bound: i $<$ length ts
  assume ts-i: ts!i $=$ (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  show $\forall$sop $\in$ write-sops sb. valid-sop sop
  using empty-sb [OF i-bound ts-i] **by** auto
qed

sublocale $\text{initial}_{\text{sb}}$ $\subseteq$ valid-store-sops
proof
  fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p
  assume i-bound: i $<$ length ts
  assume ts-i: ts!i $=$ (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  show $\forall$sop $\in$ store-sops is. valid-sop sop
  using empty-is [OF i-bound ts-i] **by** auto
qed

sublocale $\text{initial}_{\text{sb}}$ $\subseteq$ valid-sops ..

sublocale $\text{initial}_{\text{sb}} \subseteq$ valid-reads
proof
  fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p
  assume i-bound: i < length ts
  assume ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  show reads-consistent False $\mathcal{O}$ m sb
  using empty-sb [OF i-bound ts-i] **by** auto
qed

sublocale $\text{initial}_{\text{sb}} \subseteq$ valid-history
proof
  fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p
  assume i-bound: i < length ts
  assume ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  show program.history-consistent program-step j (hd-prog p sb) sb
  using empty-sb [OF i-bound ts-i] **by** (auto simp add: program.history-consistent.simps)
qed

sublocale $\text{initial}_{\text{sb}} \subseteq$ valid-data-dependency
proof
  fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p
  assume i-bound: i < length ts
  assume ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  show data-dependency-consistent-instrs (dom j) is
  using empty-is [OF i-bound ts-i] **by** auto
next
  fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p
  assume i-bound: i < length ts
  assume ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  show load-tmps is $\cap \bigcup$(fst ‘ write-sops sb) = {}
  using empty-is [OF i-bound ts-i] empty-sb [OF i-bound ts-i] **by** auto
qed

sublocale $\text{initial}_{\text{sb}} \subseteq$ load-tmps-fresh
proof
  fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p
  assume i-bound: i < length ts
  assume ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  show load-tmps is $\cap$ dom j = {}
  using empty-is [OF i-bound ts-i] **by** auto
qed

sublocale $\text{initial}_{\text{sb}} \subseteq$ enough-flushs
proof
  fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p
  assume i-bound: i < length ts
  assume ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  show outstanding-refs is-volatile-Write_{sb} sb = {}
  using empty-sb [OF i-bound ts-i] **by** auto
qed

sublocale $\text{initial}_{\text{sb}} \subseteq$ valid-program-history
proof
  fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p sb_1 sb_2
  assume i-bound: i < length ts
  assume ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  assume sb: sb=sb_1@sb_2
  show $\exists$isa. instrs sb_2 @ is = isa @ prog-instrs sb_2
  using empty-sb [OF i-bound ts-i] empty-is [OF i-bound ts-i] sb **by** auto
next

```
  fix i is 𝒪 ℛ 𝒟 j sb p
  assume i-bound: i < length ts
  assume ts-i: ts!i = (p,is,j,sb,𝒟,𝒪,ℛ)
  show last-prog p sb = p
  using empty-sb [OF i-bound ts-i] by auto
qed


inductive
 sim-config:: ('p,'p store-buffer,bool,owns,rels) thread-config list × memory × shared ⇒
             ('p, unit,bool,owns,rels) thread-config list × memory × shared  ⇒ bool
 (‹- ∼ -› [60,60] 100)
where
 ⟦m = flush-all-until-volatile-write ts_sb m_sb;
  𝒮 = share-all-until-volatile-write ts_sb 𝒮_sb;
  length ts_sb = length ts;
  ∀i < length ts_sb.
       let (p, is_sb, j, sb, 𝒟_sb, 𝒪, ℛ) = ts_sb!i;
           suspends = dropWhile (Not ∘ is-volatile-Write_sb) sb
        in  ∃is 𝒟. instrs suspends @ is_sb = is @ prog-instrs suspends ∧
                𝒟_sb = (𝒟 ∨ outstanding-refs is-volatile-Write_sb sb ≠ {}) ∧
            ts!i = (hd-prog p suspends,
                   is,
                   j |` (dom j − read-tmps suspends),(),
                   𝒟,
                   acquired True (takeWhile (Not ∘ is-volatile-Write_sb) sb) 𝒪,
                   release (takeWhile (Not ∘ is-volatile-Write_sb) sb) (dom 𝒮_sb) ℛ )
 ⟧
  ⟹
   (ts_sb,m_sb,𝒮_sb) ∼ (ts,m,𝒮)
```

The machine without history only stores writes in the store-buffer.

```
inductive
sim-history-config::
 ('p,'p store-buffer,'dirty,'owns,'rels) thread-config list ⇒ ('p,'p store-buffer,bool,owns,rels) thread-config list
⇒ bool
 (‹- ∼_h - › [60,60] 100)
where
 ⟦length ts = length ts_h;
  ∀i < length ts.
      (∃𝒪' 𝒟' ℛ'.
        let (p,is, j, sb,𝒟, 𝒪,ℛ) = ts_h!i in
             ts!i=(p,is, j, filter is-Write_sb sb,𝒟',𝒪',ℛ') ∧
             (filter is-Write_sb sb = [] ⟶ sb=[]))
 ⟧
  ⟹
   ts ∼_h ts_h

lemma (in initial_sb) history-refl:ts ∼_h ts
apply −
apply (rule sim-history-config.intros)
apply  simp
apply clarsimp
subgoal for i
apply (case-tac ts!i)
apply (drule-tac  i=i in empty-sb)
apply  assumption
apply auto
done
done

lemma share-all-empty: ∀i p is xs sb 𝒟 𝒪 ℛ. i < length ts ⟶ ts!i=(p,is,xs,sb,𝒟,𝒪,ℛ)⟶ sb=[]
  ⟹ share-all-until-volatile-write ts 𝒮 = 𝒮
```

399

```
apply (induct ts)
apply  clarsimp
apply clarsimp
apply (frule-tac x=0 in spec)
apply clarsimp
apply force
done

lemma flush-all-empty: ∀i p is xs sb 𝒟 𝒪 ℛ. i < length ts ⟶ ts!i=(p,is,xs,sb,𝒟,𝒪,ℛ)⟶ sb=[]
  ⟹ flush-all-until-volatile-write ts m = m
apply (induct ts)
apply  clarsimp
apply clarsimp
apply (frule-tac x=0 in spec)
apply clarsimp
apply force
done

lemma sim-config-emptyE:
  assumes empty:
  ∀i p is xs sb 𝒟 𝒪 ℛ. i < length ts_sb ⟶ ts_sb!i=(p,is,xs,sb,𝒟,𝒪,ℛ)⟶ sb=[]
  assumes sim: (ts_sb,m_sb,𝒮_sb) ∼ (ts,m,𝒮)
  shows 𝒮 = 𝒮_sb ∧ m = m_sb ∧ length ts = length ts_sb ∧
        (∀i < length ts_sb.
          let (p, is, j, sb, 𝒟, 𝒪, ℛ) = ts_sb!i
           in ts!i = (p, is, j, (), 𝒟, 𝒪, ℛ))
proof −
  from sim
  show ?thesis
  apply cases
  apply (clarsimp simp add: flush-all-empty [OF empty] share-all-empty [OF empty])
  subgoal for i
  apply (drule-tac x=i in spec)
  apply (cut-tac i=i in empty [rule-format])
  apply clarsimp
  apply assumption
  apply (auto simp add: Let-def)
  done
  done
qed

lemma sim-config-emptyI:
  assumes empty:
  ∀i p is xs sb 𝒟 𝒪 ℛ. i < length ts_sb ⟶ ts_sb!i=(p,is,xs,sb,𝒟,𝒪,ℛ)⟶ sb=[]
  assumes leq: length ts = length ts_sb
  assumes ts: (∀i < length ts_sb.
          let (p, is, j, sb, 𝒟, 𝒪, ℛ) = ts_sb!i
           in ts!i = (p, is, j, (), 𝒟, 𝒪, ℛ))
  shows (ts_sb,m_sb,𝒮_sb) ∼ (ts,m_sb,𝒮_sb)
apply (rule sim-config.intros)
apply    (simp add: flush-all-empty [OF empty])
apply   (simp add: share-all-empty [OF empty])
apply  (simp add: leq)
apply (clarsimp)
apply (frule (1) empty [rule-format])
using ts
apply (auto simp add: Let-def)
done
lemma mem-eq-un-eq: ⟦length ts'=length ts; ∀i< length ts'. P (ts'!i) = Q (ts!i) ⟧ ⟹ (⋃x∈set ts'. P x) =
(⋃x∈set ts. Q x)
apply (auto simp add: in-set-conv-nth )
```

```
apply (force dest!: nth-mem)
apply (frule nth-mem)
subgoal for x i
apply (drule-tac x=i in spec)
apply auto
done
done


lemma (in program) trace-to-steps:
assumes trace: trace c 0 k
shows steps: c 0 ⇒d* c k
using trace
proof (induct k)
  case 0
  show c 0 ⇒d* c 0
    by auto
next
  case (Suc k)
  have prem: trace c 0 (Suc k) by fact
  hence trace c 0 k
    by (auto simp add: program-trace-def)
  from Suc.hyps [OF this]
  have c 0 ⇒d* c k .
  also
  term program-trace
  from prem interpret program-trace program-step  c 0 Suc k .
  from step [of k] have c (k) ⇒d c (Suc k)
    by auto
  finally show ?case .
qed

lemma (in program) safe-reach-to-safe-reach-upto:
  assumes safe-reach: safe-reach-direct safe c0
  shows safe-reach-upto n safe c0
proof
  fix k c l
  assume k-n: k ≤ n
  assume trace: trace c 0 k
  assume c-0: c 0 = c0
  assume l-k: l ≤ k
  show safe (c l)
  proof -
    from trace k-n l-k have trace': trace c 0 l
      by (auto simp add: program-trace-def)
    from trace-to-steps [OF trace']
    have c 0 ⇒d* c l.
    with safe-reach c-0 show safe (c l)
    by (cases c l) (auto simp add: safe-reach-def)
  qed
qed

lemma (in program-progress) safe-free-flowing-implies-safe-delayed':
  assumes init: initialsb tssb 𝒮sb
  assumes sim: (tssb,msb,𝒮sb) ∼ (ts,m,𝒮)
  assumes safe-reach-ff: safe-reach-direct safe-free-flowing (ts,m,𝒮)
  shows safe-reach-direct safe-delayed (ts,m,𝒮)
proof -
  from init
  interpret ini: initialsb tssb 𝒮sb .
  from sim obtain
```

m: m = flush-all-until-volatile-write ts_{sb} m_{sb} **and**
$\mathcal{S}$: $\mathcal{S}$ = share-all-until-volatile-write ts_{sb} $\mathcal{S}_{sb}$ **and**
leq: length ts_{sb} = length ts **and**
t-sim: $\forall$ i < length ts_{sb}.
let (p, is_{sb}, j, sb, $\mathcal{D}_{sb}$, $\mathcal{O}$, $\mathcal{R}$) = ts_{sb}!i;
suspends = dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb
in $\exists$ is $\mathcal{D}$. instrs suspends @ is_{sb} = is @ prog-instrs suspends $\wedge$
$\mathcal{D}_{sb}$ = ($\mathcal{D}$ $\vee$ outstanding-refs is-volatile-$Write_{sb}$ sb $\neq$ {}) $\wedge$
ts!i = (hd-prog p suspends,
is,
j |' (dom j − read-tmps suspends),(),
$\mathcal{D}$,
acquired True (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb) $\mathcal{O}$,
release (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb) (dom $\mathcal{S}_{sb}$) $\mathcal{R}$)
by cases auto

from ini.empty-sb
have shared-eq: $\mathcal{S}$ = $\mathcal{S}_{sb}$
apply (simp only: $\mathcal{S}$)
apply (rule share-all-empty)
apply force
done
have sd: simple-ownership-distinct ts
proof
fix i j p_i is_i $\mathcal{O}_i$ $\mathcal{R}_i$ $\mathcal{D}_i$ j_i sb_i p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j
assume i-bound: i < length ts **and**
j-bound: j < length ts **and**
neq-i-j: i $\neq$ j **and**
ts-i: ts ! i = (p_i, is_i, j_i, sb_i, $\mathcal{D}_i$, $\mathcal{O}_i$, $\mathcal{R}_i$) **and**
ts-j: ts ! j = (p_j, is_j, j_j, sb_j, $\mathcal{D}_j$, $\mathcal{O}_j$, $\mathcal{R}_j$)
show ($\mathcal{O}_i$) $\cap$ ($\mathcal{O}_j$) = {}
proof −
from t-sim [simplified leq, rule-format, OF i-bound] ini.empty-sb [simplified leq, OF i-bound]
have ts-i: ts_{sb}!i = (p_i,is_i,j_i,[],$\mathcal{D}_i$,$\mathcal{O}_i$,$\mathcal{R}_i$)
using ts-i
by (force simp add: Let-def)
from t-sim [simplified leq, rule-format, OF j-bound] ini.empty-sb [simplified leq, OF j-bound]
have ts-j: ts_{sb}!j = (p_j,is_j,j_j,[],$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
using ts-j
by (force simp add: Let-def)
from ini.simple-ownership-distinct [simplified leq, OF i-bound j-bound neq-i-j ts-i ts-j]
show ?thesis **.**
qed
qed
have ro: read-only-unowned $\mathcal{S}$ ts
proof
fix i is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p
assume i-bound: i < length ts
assume ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
show $\mathcal{O}$ $\cap$ read-only $\mathcal{S}$ = {}
proof −
from t-sim [simplified leq, rule-format, OF i-bound] ini.empty-sb [simplified leq, OF i-bound]
have ts-i: ts_{sb}!i = (p,is,j,[],$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
using ts-i
by (force simp add: Let-def)
from ini.read-only-unowned [simplified leq, OF i-bound ts-i] shared-eq
show ?thesis **by** simp
qed
qed
have us: unowned-shared $\mathcal{S}$ ts
proof

```
    show − (⋃((λ(-, -, -, -, -, 𝒪, -). 𝒪) ‘ set ts)) ⊆ dom 𝒮
    proof −
      have (⋃((λ(-, -, -, -, -, 𝒪, -). 𝒪) ‘ set ts_sb)) = (⋃((λ(-, -, -, -, -, 𝒪, -). 𝒪) ‘ set ts))
        apply clarsimp
        apply (rule mem-eq-un-eq)
        apply (simp add: leq)
        apply clarsimp
        apply (frule t-sim [rule-format])
        apply (clarsimp simp add: Let-def)
        apply (drule (1) ini.empty-sb)
        apply auto
        done
      with ini.unowned-shared show ?thesis by (simp only: shared-eq)
    qed
  qed
  {
    fix i is 𝒪 ℛ 𝒟 j sb p
    assume i-bound: i < length ts
    assume ts-i: ts!i = (p,is,j,sb,𝒟,𝒪,ℛ)
    have ℛ = Map.empty
    proof −
      from t-sim [simplified leq, rule-format, OF i-bound] ini.empty-sb [simplified leq, OF i-bound]
      have ts-i: ts_sb!i = (p,is,j,[],𝒟,𝒪,ℛ)
      using ts-i
        by (force simp add: Let-def)
      from ini.empty-rels [simplified leq, OF i-bound ts-i]
      show ?thesis .
    qed
  }
  with us have initial: initial (ts, m, 𝒮)
    by (fastforce simp add: initial-def)

  {
    fix ts′ 𝒮′ m′
    assume steps: (ts,m,𝒮) ⇒_d^* (ts′,m′,𝒮′)
    have safe-delayed (ts′,m′,𝒮′)
    proof −
      from steps-to-trace [OF steps] obtain c k
      where trace: trace c 0 k and c-0: c 0 = (ts,m,𝒮) and c-k: c k = (ts′,m′,𝒮′)
        by auto
      from safe-reach-to-safe-reach-upto [OF safe-reach-ff]
      have safe-upto-k: safe-reach-upto k safe-free-flowing (ts, m, 𝒮).
      from safe-free-flowing-implies-safe-delayed [OF - - - - safe-upto-k, simplified, OF initial sd ro us]
      have safe-reach-upto k safe-delayed (ts, m, 𝒮).
      then interpret program-safe-reach-upto program-step k safe-delayed (ts,m,𝒮) .
      from safe-config [where c=c and k=k and l=k, OF - trace c-0] c-k show ?thesis by simp
    qed
  }
  then show ?thesis
    by (clarsimp simp add: safe-reach-def)
qed


lemma map-onws-sb-owned:⋀j. j < length ts ⟹ map 𝒪-sb ts ! j = (𝒪_j,sb_j) ⟹ map owned ts ! j = 𝒪_j
apply (induct ts)
apply  simp
subgoal for t ts j
apply (case-tac j)
apply  (case-tac t)
apply  auto
done
```

done

lemma map-onws-sb-owned′:$\bigwedge$j. j < length ts $\Longrightarrow$ $\mathcal{O}$-sb (ts ! j) = ($\mathcal{O}_j$,sb_j) $\Longrightarrow$ owned (ts ! j) = $\mathcal{O}_j$
apply (induct ts)
apply simp
subgoal for t ts j
apply (case-tac j)
apply (case-tac t)
apply auto
done
done

lemma read-only-read-acquired-unforwarded-acquire-witness:
 $\bigwedge\mathcal{S}$ $\mathcal{O}$ X.⟦non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb;
 sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb; a $\notin$ read-only $\mathcal{S}$; a $\notin$ $\mathcal{O}$;
 a $\in$ unforwarded-non-volatile-reads sb X⟧
$\Longrightarrow$($\exists$sop a′ v ys zs A L R W.
  sb = ys @ Write_{sb} True a′ sop v A L R W # zs $\wedge$
  a $\in$ A $\wedge$ a $\notin$ outstanding-refs is-Write_{sb} ys $\wedge$ a′ $\neq$ a) $\vee$
($\exists$A L R W ys zs. sb = ys @ Ghost_{sb} A L R W# zs $\wedge$ a $\in$ A $\wedge$ a $\notin$ outstanding-refs is-Write_{sb} ys)
proof (induct sb)
 case Nil **thus** ?case **by** simp
next
 case (Cons x sb)
 show ?case
 proof (cases x)
  case (Write_{sb} volatile a′ sop v A L R W)
  show ?thesis
  proof (cases volatile)
   case True
   note volatile=this
   from Cons.prems **obtain**
 nvo′: non-volatile-owned-or-read-only True ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) ($\mathcal{O}$ $\cup$ A − R) sb **and**
 a-nro: a $\notin$ read-only $\mathcal{S}$ **and**
 a-unowned: a $\notin$ $\mathcal{O}$ **and**
 A-shared-owns: A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}$ **and** L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and**
 R-owns: R $\subseteq$ $\mathcal{O}$ **and**
 consis′: sharing-consistent ($\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) ($\mathcal{O}$ $\cup$ A − R) sb **and**
 a-unforw: a $\in$ unforwarded-non-volatile-reads sb (insert a′ X)
 by (clarsimp simp add: Write_{sb} True)
   from unforwarded-not-written [OF a-unforw]
   have a-notin: a $\notin$ insert a′ X**.**
   hence a′-a: a′ $\neq$ a
    by simp
   from R-owns a-unowned
   have a-R: a $\notin$ R
 by auto
   show ?thesis
   proof (cases a $\in$ A)
 case True
 then show ?thesis
  apply −
  apply (rule disjI1)
  apply (rule-tac x=sop **in** exI)
  apply (rule-tac x=a′ **in** exI)
  apply (rule-tac x=v **in** exI)
  apply (rule-tac x=[] **in** exI)
  apply (rule-tac x=sb **in** exI)
  apply (simp add: Write_{sb} volatile True a′-a)

```
  done
     next
case False
with a-unowned R-owns a-nro L-A A-R
obtain a-nro': a ∉ read-only (𝒮 ⊕W R ⊖A L) and a-unowned': a ∉ 𝒪 ∪ A − R
  by (force simp add: in-read-only-convs)

from Cons.hyps [OF nvo' consis' a-nro' a-unowned' a-unforw]
have (∃sop a' v ys zs A L R W.
                sb = ys @ Writesb True a' sop v A L R W # zs ∧
                a ∈ A ∧ a ∉ outstanding-refs is-Writesb ys ∧ a' ≠ a) ∨
             (∃A L R W ys zs. sb = ys @ Ghostsb A L R W# zs ∧ a ∈ A ∧ a ∉ outstanding-refs is-Writesb ys)
             (is ?write ∨ ?ghst)
  by simp
then show ?thesis
       proof
  assume ?write

  then obtain sop' a'' v' ys zs A' L' R' W' where
          sb: sb = ys @ Writesb True a'' sop' v' A' L' R' W' # zs and
          props: a ∈ A' a ∉ outstanding-refs is-Writesb ys ∧ a'' ≠ a
    by auto


  show ?thesis
  using props False a-notin sb
    apply −
    apply (rule disjI1)
    apply (rule-tac x=sop' in exI)
    apply (rule-tac x=a'' in exI)
    apply (rule-tac x=v' in exI)
    apply (rule-tac x=(x#ys) in exI)
    apply (rule-tac x=zs in exI)
    apply (simp add: Writesb volatile False a'-a)
    done
next
  assume ?ghst
  then obtain ys zs A' L' R' W'  where
          sb: sb = ys @ Ghostsb A' L' R' W'# zs and
          props: a ∈ A' a ∉ outstanding-refs is-Writesb ys
    by auto


  show ?thesis
  using props False a-notin sb
    apply −
    apply (rule disjI2)
    apply (rule-tac x=A' in exI)
    apply (rule-tac x=L' in exI)
    apply (rule-tac x=R' in exI)
    apply (rule-tac x=W' in exI)
    apply (rule-tac x=(x#ys) in exI)
    apply (rule-tac x=zs in exI)
    apply (simp add: Writesb volatile False a'-a)
    done
qed
     qed
   next
     case False
     from Cons.prems obtain
consis': sharing-consistent 𝒮 𝒪 sb and
```

```
a-nro′: a ∉ read-only 𝒮 and
a-unowned: a ∉ 𝒪 and
a-ro′: a′ ∈ 𝒪 and
nvo′: non-volatile-owned-or-read-only True 𝒮 𝒪 sb and
a-unforw′: a ∈ unforwarded-non-volatile-reads sb (insert a′ X)
by (auto simp add: Write_sb False split: if-split-asm)

     from unforwarded-not-written [OF a-unforw′]
     have a-notin: a ∉ insert a′ X.

     from Cons.hyps [OF nvo′ consis′ a-nro′ a-unowned a-unforw′]
     have (∃sop a′ v ys zs A L R W.
                 sb = ys @ Write_sb True a′ sop v A L R W # zs ∧
                 a ∈ A ∧ a ∉ outstanding-refs is-Write_sb ys ∧ a′ ≠ a) ∨
              (∃A L R W ys zs. sb = ys @ Ghost_sb A L R W# zs ∧ a ∈ A ∧ a ∉ outstanding-refs is-Write_sb ys)
       (is ?write ∨ ?ghst)
by simp
then show ?thesis
       proof
  assume ?write

  then obtain sop′ a′′ v′ ys zs A′ L′ R′ W′ where
          sb: sb = ys @ Write_sb True a′′ sop′ v′ A′ L′ R′ W′ # zs and
          props: a ∈ A′ a ∉ outstanding-refs is-Write_sb ys ∧ a′′ ≠ a
    by auto


  show ?thesis
  using props False a-notin sb
    apply −
    apply (rule disjI1)
    apply (rule-tac x=sop′ in exI)
    apply (rule-tac x=a′′ in exI)
    apply (rule-tac x=v′ in exI)
    apply (rule-tac x=(x#ys) in exI)
    apply (rule-tac x=zs in exI)
    apply (simp add: Write_sb False )
    done
next
  assume ?ghst
  then obtain ys zs A′ L′ R′ W′  where
          sb: sb = ys @ Ghost_sb A′ L′ R′ W′ # zs and
          props: a ∈ A′ a ∉ outstanding-refs is-Write_sb ys
    by auto


  show ?thesis
  using props False a-notin sb
    apply −
    apply (rule disjI2)
    apply (rule-tac x=A′ in exI)
    apply (rule-tac x=L′ in exI)
    apply (rule-tac x=R′ in exI)
    apply (rule-tac x=W′ in exI)
    apply (rule-tac x=(x#ys) in exI)
    apply (rule-tac x=zs in exI)
    apply (simp add: Write_sb False )
    done
qed
     qed
   next
```

case (Read$_{sb}$ volatile a′ t v)
from Cons.prems
obtain
consis′: sharing-consistent $\mathcal{S}$ $\mathcal{O}$ sb **and**
a-nro′: a $\notin$ read-only $\mathcal{S}$ **and**
a-unowned: a $\notin$ $\mathcal{O}$ **and**
nvo′: non-volatile-owned-or-read-only True $\mathcal{S}$ $\mathcal{O}$ sb **and**
a-unforw: a $\in$ unforwarded-non-volatile-reads sb X
by (auto simp add: Read$_{sb}$ split: if-split-asm)
from Cons.hyps [OF nvo′ consis′ a-nro′ a-unowned a-unforw]
have ($\exists$ sop a′ v ys zs A L R W.
sb = ys @ Write$_{sb}$ True a′ sop v A L R W # zs $\wedge$
a $\in$ A $\wedge$ a $\notin$ outstanding-refs is-Write$_{sb}$ ys $\wedge$ a′ $\neq$ a) $\vee$
($\exists$ A L R W ys zs. sb = ys @ Ghost$_{sb}$ A L R W# zs $\wedge$ a $\in$ A $\wedge$ a $\notin$ outstanding-refs is-Write$_{sb}$ ys)
(**is** ?write $\vee$?ghst)
by simp
then show ?thesis
proof
assume ?write

then obtain sop′ a′′ v′ ys zs A′ L′ R′ W′ **where**
sb: sb = ys @ Write$_{sb}$ True a′′ sop′ v′ A′ L′ R′ W′ # zs **and**
props: a $\in$ A′ a $\notin$ outstanding-refs is-Write$_{sb}$ ys $\wedge$ a′′ $\neq$ a
by auto

show ?thesis
using props sb
apply −
apply (rule disjI1)
apply (rule-tac x=sop′ **in** exI)
apply (rule-tac x=a′′ **in** exI)
apply (rule-tac x=v′ **in** exI)
apply (rule-tac x=(x#ys) **in** exI)
apply (rule-tac x=zs **in** exI)
apply (simp add: Read$_{sb}$)
done
next
assume ?ghst
then obtain ys zs A′ L′ R′ W′ **where**
sb: sb = ys @ Ghost$_{sb}$ A′ L′ R′ W′# zs **and**
props: a $\in$ A′ a $\notin$ outstanding-refs is-Write$_{sb}$ ys
by auto

show ?thesis
using props sb
apply −
apply (rule disjI2)
apply (rule-tac x=A′ **in** exI)
apply (rule-tac x=L′ **in** exI)
apply (rule-tac x=R′ **in** exI)
apply (rule-tac x=W′ **in** exI)
apply (rule-tac x=(x#ys) **in** exI)
apply (rule-tac x=zs **in** exI)
apply (simp add: Read$_{sb}$)
done
qed
next
case Prog$_{sb}$
from Cons.prems

```
  obtain
    consis′: sharing-consistent 𝒮 𝒪 sb and
    a-nro′: a ∉ read-only 𝒮 and
    a-unowned: a ∉ 𝒪 and
    nvo′: non-volatile-owned-or-read-only True 𝒮 𝒪 sb and
    a-unforw: a ∈ unforwarded-non-volatile-reads sb X
    by (auto simp add: Prog_sb)
  from Cons.hyps [OF nvo′ consis′ a-nro′ a-unowned a-unforw]
  have (∃sop a′ v ys zs A L R W.
            sb = ys @ Write_sb True a′ sop v A L R W # zs ∧
            a ∈ A ∧ a ∉ outstanding-refs is-Write_sb ys ∧ a′ ≠ a) ∨
         (∃A L R W ys zs. sb = ys @ Ghost_sb A L R W# zs ∧ a ∈ A ∧ a ∉ outstanding-refs is-Write_sb ys)
    (is ?write ∨ ?ghst)
    by simp
  then show ?thesis
  proof
    assume ?write

    then obtain sop′ a′′ v′ ys zs A′ L′ R′ W′ where
      sb: sb = ys @ Write_sb True a′′ sop′ v′ A′ L′ R′ W′ # zs and
      props: a ∈ A′ a ∉ outstanding-refs is-Write_sb ys ∧ a′′ ≠ a
      by auto


    show ?thesis
    using props sb
      apply −
apply (rule disjI1)
apply (rule-tac x=sop′ in exI)
apply (rule-tac x=a′′ in exI)
apply (rule-tac x=v′ in exI)
apply (rule-tac x=(x#ys) in exI)
apply (rule-tac x=zs in exI)
apply (simp add: Prog_sb)
done
  next
    assume ?ghst
    then obtain ys zs A′ L′ R′ W′ where
      sb: sb = ys @ Ghost_sb A′ L′ R′ W′# zs and
      props: a ∈ A′ a ∉ outstanding-refs is-Write_sb ys
      by auto


    show ?thesis
    using props sb
    apply −
    apply (rule disjI2)
    apply (rule-tac x=A′ in exI)
    apply (rule-tac x=L′ in exI)
    apply (rule-tac x=R′ in exI)
    apply (rule-tac x=W′ in exI)
    apply (rule-tac x=(x#ys) in exI)
    apply (rule-tac x=zs in exI)
    apply (simp add: Prog_sb )
    done
  qed
next
  case (Ghost_sb A L R W)
  from Cons.prems obtain
    nvo′: non-volatile-owned-or-read-only True (𝒮 ⊕_W R ⊖_A L) (𝒪 ∪ A − R) sb and
    a-nro: a ∉ read-only 𝒮 and
```

```
    a-unowned: a ∉ 𝒪 and
    A-shared-owns: A ⊆ dom 𝒮 ∪ 𝒪 and L-A: L ⊆ A and A-R: A ∩ R = {} and
    R-owns: R ⊆ 𝒪 and
    consis′: sharing-consistent (𝒮 ⊕W R ⊖A L) (𝒪 ∪ A − R) sb and
    a-unforw: a ∈ unforwarded-non-volatile-reads sb X
    by (clarsimp simp add: Ghost_sb)

  show ?thesis
  proof (cases a ∈ A)
    case True
    then show ?thesis
      apply −
apply (rule disjI2)
apply (rule-tac x=A in exI)
apply (rule-tac x=L in exI)
apply (rule-tac x=R in exI)
apply (rule-tac x=W in exI)
apply (rule-tac x=[] in exI)
apply (rule-tac x=sb in exI)
apply (simp add: Ghost_sb True)
done
  next
    case False

    with a-unowned a-nro L-A R-owns a-nro L-A A-R
    obtain a-nro′: a ∉ read-only (𝒮 ⊕W R ⊖A L) and a-unowned′: a ∉ 𝒪 ∪ A − R
by (force simp add: in-read-only-convs)
    from Cons.hyps [OF nvo′ consis′ a-nro′ a-unowned′ a-unforw]
    have (∃sop a′ v ys zs A L R W.
            sb = ys @ Write_sb True a′ sop v A L R W # zs ∧
            a ∈ A ∧ a ∉ outstanding-refs is-Write_sb ys ∧ a′ ≠ a) ∨
          (∃A L R W ys zs. sb = ys @ Ghost_sb A L R W# zs ∧ a ∈ A ∧ a ∉ outstanding-refs is-Write_sb ys)
      (is ?write ∨ ?ghst)
by simp
    then show ?thesis
    proof
assume ?write

then obtain sop′ a′′ v′ ys zs A′ L′ R′ W′ where
      sb: sb = ys @ Write_sb True a′′ sop′ v′ A′ L′ R′ W′ # zs and
      props: a ∈ A′ a ∉ outstanding-refs is-Write_sb ys ∧ a′′ ≠ a
  by auto


show ?thesis
using props sb
  apply −
  apply (rule disjI1)
  apply (rule-tac x=sop′ in exI)
  apply (rule-tac x=a′′ in exI)
  apply (rule-tac x=v′ in exI)
  apply (rule-tac x=(x#ys) in exI)
  apply (rule-tac x=zs in exI)
  apply (simp add: Ghost_sb False )
  done
    next
assume ?ghst
then obtain ys zs A′ L′ R′ W′  where
      sb: sb = ys @ Ghost_sb A′ L′ R′ W′# zs and
      props: a ∈ A′ a ∉ outstanding-refs is-Write_sb ys
  by auto
```

```
show ?thesis
using props sb
  apply −
  apply (rule disjI2)
  apply (rule-tac x=A′ in exI)
  apply (rule-tac x=L′ in exI)
  apply (rule-tac x=R′ in exI)
  apply (rule-tac x=W′ in exI)
  apply (rule-tac x=(x#ys) in exI)
  apply (rule-tac x=zs in exI)
  apply (simp add: Ghost_sb False )
  done
qed
    qed
  qed
 qed
```

```
lemma release-shared-exchange-weak:
assumes shared-eq: ∀ a ∈ 𝒪 ∪ all-acquired sb. (𝒮′::shared) a = 𝒮 a
assumes consis: weak-sharing-consistent 𝒪 sb
shows release sb (dom 𝒮′) ℛ = release sb (dom 𝒮) ℛ
using shared-eq consis
proof (induct sb arbitrary: 𝒮 𝒮′ 𝒪 ℛ)
  case Nil thus ?case by auto
next
  case (Cons x sb)
  show ?case
  proof (cases x)
    case (Write_sb volatile a sop v A L R W)
    show ?thesis
    proof (cases volatile)
      case True

      from Cons.prems obtain
 L-A: L ⊆ A and A-R: A ∩ R = {} and R-owns: R ⊆ 𝒪 and
 consis′: weak-sharing-consistent (𝒪 ∪ A − R) sb and
        shared-eq: ∀ a ∈ 𝒪 ∪ A ∪ all-acquired sb. 𝒮′ a = 𝒮 a
 by (clarsimp simp add: Write_sb True )
      from shared-eq
      have shared-eq′: ∀ a∈𝒪 ∪ A − R ∪ all-acquired sb. (𝒮′ ⊕_W R ⊖_A L) a = (𝒮 ⊕_W R ⊖_A L) a
        by (auto simp add: augment-shared-def restrict-shared-def)
      from Cons.hyps [OF shared-eq′ consis′]
      have release sb (dom (𝒮′ ⊕_W R ⊖_A L)) Map.empty = release sb (dom (𝒮 ⊕_W R ⊖_A L)) Map.empty .
      then show ?thesis
        by (auto  simp add: Write_sb True domIff)
    next
      case False with Cons show ?thesis
 by (auto simp add: Write_sb)
    qed
  next
    case Read_sb with Cons show ?thesis
      by auto
  next
    case Prog_sb with Cons show ?thesis
      by auto
  next
    case (Ghost_sb A L R W)
```

```
    from Cons.prems obtain
      L-A: L ⊆ A and A-R: A ∩ R = {} and R-owns: R ⊆ 𝒪 and
      consis′: weak-sharing-consistent (𝒪 ∪ A − R) sb and
      shared-eq: ∀ a ∈ 𝒪 ∪ A ∪ all-acquired sb. 𝒮′ a = 𝒮 a
      by (clarsimp simp add: Ghost_sb )
    from shared-eq
    have shared-eq′: ∀ a∈𝒪 ∪ A − R ∪ all-acquired sb. (𝒮′ ⊕_W R ⊖_A L) a = (𝒮 ⊕_W R ⊖_A L) a
      by (auto simp add: augment-shared-def restrict-shared-def)
    from shared-eq R-owns have ∀ a∈R. (a ∈ dom 𝒮) = (a ∈ dom 𝒮′)
      by (auto simp add: domIff)
    from augment-rels-shared-exchange [OF this]
    have (augment-rels (dom 𝒮′) R ℛ) = (augment-rels (dom 𝒮) R ℛ).

    with Cons.hyps [OF shared-eq′ consis′]
    have release sb (dom (𝒮′ ⊕_W R ⊖_A L)) (augment-rels (dom 𝒮′) R ℛ) =
            release sb (dom (𝒮 ⊕_W R ⊖_A L)) (augment-rels (dom 𝒮) R ℛ) by simp
    then show ?thesis
      by (clarsimp  simp add: Ghost_sb domIff)
  qed
qed


lemma read-only-share-all-shared: ⋀𝒮. ⟦ a ∈ read-only (share sb 𝒮)⟧
⟹ a ∈ read-only 𝒮 ∪ all-shared sb
proof (induct sb)
  case Nil thus ?case by simp
next
  case (Cons x sb)
  show ?case
  proof (cases x)
    case (Write_sb volatile a sop v A L R W)
    show ?thesis
    proof (cases volatile)
      case True
      with Write_sb Cons.hyps [of (𝒮 ⊕_W R ⊖_A L)] Cons.prems
      show ?thesis
        by (auto simp add: read-only-def augment-shared-def restrict-shared-def
          split: if-split-asm option.splits)
    next
      case False with Write_sb Cons show ?thesis by auto
    qed
  next
    case Read_sb with Cons show ?thesis by auto
  next
    case Prog_sb with Cons show ?thesis by auto
  next
    case (Ghost_sb A L R W)
    with Cons.hyps [of (𝒮 ⊕_W R ⊖_A L)] Cons.prems
    show ?thesis
      by (auto simp add: read-only-def augment-shared-def restrict-shared-def
          split: if-split-asm option.splits)
  qed
qed

lemma read-only-shared-all-until-volatile-write-subset′:
⋀𝒮.
read-only (share-all-until-volatile-write ts 𝒮) ⊆
  read-only 𝒮 ∪ (⋃((λ(-, -, -, sb, -, - ,-). all-shared (takeWhile (Not ∘ is-volatile-Write_sb) sb)) ‘ set ts))
proof (induct ts)
  case Nil thus ?case by simp
next
```

```
  case (Cons t ts)
  obtain p is 𝒪 ℛ 𝒟 j sb where
    t: t = (p,is,j,sb,𝒟,𝒪,ℛ)
    by (cases t)


  have aargh: (Not ∘ is-volatile-Write_sb) = (λa. ¬ is-volatile-Write_sb a)
    by (rule ext) auto


  let ?take-sb = (takeWhile (Not ∘ is-volatile-Write_sb) sb)
  let ?drop-sb = (dropWhile (Not ∘ is-volatile-Write_sb) sb)



  {
    fix a
    assume a-in: a ∈ read-only
           (share-all-until-volatile-write ts
              (share ?take-sb 𝒮)) and
    a-notin-shared: a ∉ read-only 𝒮 and
    a-notin-rest: a ∉ (⋃((λ(-, -, -, sb, -, - ,-). all-shared (takeWhile (Not ∘ is-volatile-Write_sb) sb)) ‘ set ts))
    have a ∈ all-shared (takeWhile (Not ∘ is-volatile-Write_sb) sb)
    proof −
      from Cons.hyps [of (share ?take-sb 𝒮)] a-in a-notin-rest
      have a ∈ read-only (share ?take-sb 𝒮)
        by (auto simp add: aargh)
      from read-only-share-all-shared [OF this] a-notin-shared
      show ?thesis by auto
    qed
  }

  then show ?case
    by (auto simp add: t aargh)
qed


lemma read-only-share-acquired-all-shared:
  ⋀𝒪 𝒮. weak-sharing-consistent 𝒪 sb ⟹ 𝒪 ∩ read-only 𝒮 = {} ⟹
  a ∈ read-only (share sb 𝒮) ⟹ a ∈ 𝒪 ∪ all-acquired sb ⟹ a ∈ all-shared sb
proof (induct sb)
  case Nil thus ?case by auto
next
  case (Cons x sb)
  show ?case
  proof (cases x)
    case (Write_sb volatile a′ sop v A L R W)
    show ?thesis
    proof (cases volatile)
      case True
      note volatile=this
      from Cons.prems obtain
 owns-ro: 𝒪 ∩ read-only 𝒮 = {} and L-A:  L ⊆ A and A-R: A ∩ R = {} and
 R-owns: R ⊆ 𝒪 and consis′: weak-sharing-consistent  (𝒪 ∪ A − R)  sb and
        a-share: a ∈ read-only (share sb (𝒮 ⊕_W R ⊖_A L)) and
        a-A-all: a ∈ 𝒪 ∪ A ∪ all-acquired sb
 by (clarsimp simp add: Write_sb True)

      from owns-ro A-R R-owns have owns-ro′: (𝒪 ∪ A − R) ∩ read-only (𝒮 ⊕_W R ⊖_A L) = {}
        by (auto simp add: in-read-only-convs)
      from Cons.hyps [OF consis′ owns-ro′ a-share]
```

show ?thesis
using L-A A-R R-owns owns-ro a-A-all
by (auto simp add: $\mathsf{Write_{sb}}$ volatile augment-shared-def restrict-shared-def read-only-def domIff
split: if-split-asm)
next
case False
with Cons $\mathsf{Write_{sb}}$ show ?thesis by (auto)
qed
next
case $\mathsf{Read_{sb}}$ with Cons show ?thesis by auto
next
case $\mathsf{Prog_{sb}}$ with Cons show ?thesis by auto
next
case ($\mathsf{Ghost_{sb}}$ A L R W)
from Cons.prems obtain
owns-ro: $\mathcal{O} \cap$ read-only $\mathcal{S} = \{\}$ and L-A: $L \subseteq A$ and A-R: $A \cap R = \{\}$ and
R-owns: $R \subseteq \mathcal{O}$ and consis′: weak-sharing-consistent ($\mathcal{O} \cup A - R$) sb and
a-share: $a \in$ read-only (share sb ($\mathcal{S} \oplus_W R \ominus_A L$)) and
a-A-all: $a \in \mathcal{O} \cup A \cup$ all-acquired sb
by (clarsimp simp add: $\mathsf{Ghost_{sb}}$)

from owns-ro A-R R-owns have owns-ro′: ($\mathcal{O} \cup A - R$) $\cap$ read-only ($\mathcal{S} \oplus_W R \ominus_A L$) = {}
by (auto simp add: in-read-only-convs)
from Cons.hyps [OF consis′ owns-ro′ a-share]
show ?thesis
using L-A A-R R-owns owns-ro a-A-all
by (auto simp add: $\mathsf{Ghost_{sb}}$ augment-shared-def restrict-shared-def read-only-def domIff
split: if-split-asm)
qed
qed

lemma read-only-share-unowned′: $\bigwedge\mathcal{O}\ \mathcal{S}$.
$[\![$weak-sharing-consistent $\mathcal{O}$ sb; $\mathcal{O} \cap$ read-only $\mathcal{S} = \{\}$; $a \notin \mathcal{O} \cup$ all-acquired sb; $a \in$ read-only $\mathcal{S}]\!]$
$\Longrightarrow a \in$ read-only (share sb $\mathcal{S}$)
proof (induct sb)
case Nil thus ?case by simp
next
case (Cons x sb)
show ?case
proof (cases x)
case ($\mathsf{Write_{sb}}$ volatile a′ sop v A L R W)
show ?thesis
proof (cases volatile)
case False
with Cons $\mathsf{Write_{sb}}$ show ?thesis by auto
next
case True
from Cons.prems obtain
owns-ro: $\mathcal{O} \cap$ read-only $\mathcal{S} = \{\}$ and L-A: $L \subseteq A$ and A-R: $A \cap R = \{\}$ and
R-owns: $R \subseteq \mathcal{O}$ and consis′: weak-sharing-consistent ($\mathcal{O} \cup A - R$) sb and
a-share: $a \in$ read-only $\mathcal{S}$ and
a-notin: $a \notin \mathcal{O}$ $a \notin A$ $a \notin$ all-acquired sb
by (clarsimp simp add: $\mathsf{Write_{sb}}$ True)
from owns-ro A-R R-owns have owns-ro′: ($\mathcal{O} \cup A - R$) $\cap$ read-only ($\mathcal{S} \oplus_W R \ominus_A L$) = {}
by (auto simp add: in-read-only-convs)
from a-notin have a-notin′: $a \notin \mathcal{O} \cup A - R \cup$ all-acquired sb
by auto
from a-share a-notin L-A A-R R-owns have a-ro′: $a \in$ read-only ($\mathcal{S} \oplus_W R \ominus_A L$)
by (auto simp add: read-only-def restrict-shared-def augment-shared-def)
from Cons.hyps [OF consis′ owns-ro′ a-notin′ a-ro′]
have $a \in$ read-only (share sb ($\mathcal{S} \oplus_W R \ominus_A L$))

by auto
then show ?thesis
by (auto simp add: Write_{sb} True)
qed
next
case Read_{sb} with Cons show ?thesis by auto
next
case Prog_{sb} with Cons show ?thesis by auto
next
case (Ghost_{sb} A L R W)
from Cons.prems obtain
owns-ro: $\mathcal{O} \cap$ read-only $\mathcal{S} = \{\}$ and L-A: $L \subseteq A$ and A-R: $A \cap R = \{\}$ and
R-owns: $R \subseteq \mathcal{O}$ and consis$'$: weak-sharing-consistent $(\mathcal{O} \cup A - R)$ sb and
a-share: $a \in$ read-only $\mathcal{S}$ and
a-notin: $a \notin \mathcal{O}$ $a \notin A$ $a \notin$ all-acquired sb
by (clarsimp simp add: Ghost_{sb})
from owns-ro A-R R-owns have owns-ro$'$: $(\mathcal{O} \cup A - R) \cap$ read-only $(\mathcal{S} \oplus_W R \ominus_A L) = \{\}$
by (auto simp add: in-read-only-convs)
from a-notin have a-notin$'$: $a \notin \mathcal{O} \cup A - R \cup$ all-acquired sb
by auto
from a-share a-notin L-A A-R R-owns have a-ro$'$: $a \in$ read-only $(\mathcal{S} \oplus_W R \ominus_A L)$
by (auto simp add: read-only-def restrict-shared-def augment-shared-def)
from Cons.hyps [OF consis$'$ owns-ro$'$ a-notin$'$ a-ro$'$]
have $a \in$ read-only (share sb $(\mathcal{S} \oplus_W R \ominus_A L)$)
by auto
then show ?thesis
by (auto simp add: Ghost_{sb})
qed
qed

lemma release-False-mono:
$\bigwedge S\ \mathcal{R}.\ \mathcal{R}$ a = Some False $\Longrightarrow$ outstanding-refs is-volatile-Write_{sb} sb = {} $\Longrightarrow$
release sb S $\mathcal{R}$ a = Some False
proof (induct sb)
case Nil thus ?case by simp
next
case (Cons x sb)
show ?case
proof (cases x)
case (Ghost_{sb} A L R W)
have rels-a: $\mathcal{R}$ a = Some False by fact
then have (augment-rels S R $\mathcal{R}$) a = Some False
by (auto simp add: augment-rels-def)
from Cons.hyps [where $\mathcal{R}$ = (augment-rels S R $\mathcal{R}$), OF this] Cons.prems
show ?thesis
by (clarsimp simp add: Ghost_{sb})
next
case Write_{sb} with Cons show ?thesis by auto
next
case Read_{sb} with Cons show ?thesis by auto
next
case Prog_{sb} with Cons show ?thesis by auto
qed
qed

lemma release-False-mono-take:
$\bigwedge S\ \mathcal{R}.\ \mathcal{R}$ a = Some False $\Longrightarrow$ release (takeWhile (Not $\circ$ is-volatile-Write_{sb}) sb) S $\mathcal{R}$ a = Some False
proof (induct sb)

```
  case Nil thus ?case by simp
next
  case (Cons x sb)
  show ?case
  proof (cases x)
    case (Ghost_sb A L R W)
    have rels-a: ℛ a = Some False by fact
    then have (augment-rels S R ℛ) a = Some False
      by (auto simp add: augment-rels-def)
    from Cons.hyps [where ℛ = (augment-rels S R ℛ), OF this]
    show ?thesis
      by (clarsimp simp add: Ghost_sb)
  next
    case Write_sb with Cons show ?thesis by auto
  next
    case Read_sb with Cons show ?thesis by auto
  next
    case Prog_sb with Cons show ?thesis by auto
  qed
qed


lemma shared-switch:
  ⋀𝒮 𝒪. ⟦weak-sharing-consistent 𝒪 sb; read-only 𝒮 ∩ 𝒪 = {};
    𝒮 a ≠ Some False; share sb 𝒮 a = Some False⟧
  ⟹ a ∈ 𝒪 ∪ all-acquired sb
proof (induct sb)
  case Nil thus ?case by (auto simp add: read-only-def)
next
  case (Cons x sb)
  have aargh: (Not ∘ is-volatile-Write_sb) = (λa. ¬ is-volatile-Write_sb a)
    by (rule ext) auto
  show ?case
  proof (cases x)
    case (Ghost_sb A L R W)
    from Cons.prems obtain
      dist: read-only 𝒮 ∩ 𝒪 = {} and
      share: 𝒮 a ≠ Some False and
      share′: share sb (𝒮 ⊕_W R ⊖_A L) a = Some False and
      L-A: L ⊆ A and A-R: A ∩ R = {} and R-owns: R ⊆ 𝒪 and
      consis′: weak-sharing-consistent (𝒪 ∪ A − R) sb by (clarsimp simp add: Ghost_sb aargh)

    from dist L-A A-R R-owns have dist′:  read-only (𝒮 ⊕_W R ⊖_A L) ∩ (𝒪 ∪ A − R)= {}
      by (auto simp add: in-read-only-convs)

    show ?thesis
    proof (cases (𝒮 ⊕_W R ⊖_A L) a = Some False)
      case False
      from Cons.hyps [OF consis′ dist′ this share′]
      show ?thesis by (auto simp add: Ghost_sb)
    next
      case True
      with share L-A A-R R-owns dist
      have a ∈ 𝒪 ∪ A
        by (cases 𝒮 a)
           (auto simp add: augment-shared-def restrict-shared-def read-only-def split: if-split-asm )
      thus ?thesis by (auto simp add: Ghost_sb)
   qed
 next
   case (Write_sb volatile a′ sop v A L R W)
   show ?thesis
```

proof (cases volatile)
  case True
  note volatile=this
  from Cons.prems **obtain**
    dist: read-only $\mathcal{S} \cap \mathcal{O} = \{\}$ **and**
    share: $\mathcal{S}$ a $\neq$ Some False **and**
    share′: share sb ($\mathcal{S} \oplus_W$ R $\ominus_A$ L) a = Some False **and**
    L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq \mathcal{O}$ **and**
    consis′: weak-sharing-consistent ($\mathcal{O} \cup$ A − R) sb **by** (clarsimp simp add: Write_{sb} True aargh)

  from dist L-A A-R R-owns **have** dist′: read-only ($\mathcal{S} \oplus_W$ R $\ominus_A$ L) $\cap$ ($\mathcal{O} \cup$ A − R)= {}
    by (auto simp add: in-read-only-convs)

  show ?thesis
  proof (cases ($\mathcal{S} \oplus_W$ R $\ominus_A$ L) a = Some False)
    case False
    from Cons.hyps [OF consis′ dist′ this share′]
    show ?thesis **by** (auto simp add: Write_{sb} True)
  next
    case True
    with share L-A A-R R-owns dist
    have a $\in \mathcal{O} \cup$ A
      by (cases $\mathcal{S}$ a)
        (auto simp add: augment-shared-def restrict-shared-def read-only-def split: if-split-asm)
    thus ?thesis **by** (auto simp add: Write_{sb} volatile)
  qed
next
  case False
  with Cons **show** ?thesis **by** (auto simp add: Write_{sb})
qed
next
  case Read_{sb} **with** Cons **show** ?thesis **by** (auto)
next
  case Prog_{sb} **with** Cons **show** ?thesis **by** (auto)
qed
qed

lemma shared-switch-release-False:
  $\bigwedge\mathcal{S}\ \mathcal{R}.$ ⟦
    outstanding-refs is-volatile-Write_{sb} sb = {};
    a $\notin$ dom $\mathcal{S}$;
    a $\in$ dom (share sb $\mathcal{S}$)⟧
  $\Longrightarrow$
    release sb (dom $\mathcal{S}$) $\mathcal{R}$ a = Some False
proof (induct sb)
  case Nil **thus** ?case **by** (auto simp add: read-only-def)
next
  case (Cons x sb)
  have aargh: (Not $\circ$ is-volatile-Write_{sb}) = (λa. $\neg$ is-volatile-Write_{sb} a)
    by (rule ext) auto
  show ?case
  proof (cases x)
    case (Ghost_{sb} A L R W)
    from Cons.prems **obtain**
      a-notin: a $\notin$ dom $\mathcal{S}$ **and**
      share: a $\in$ dom (share sb ($\mathcal{S} \oplus_W$ R $\ominus_A$ L)) **and**
      out′: outstanding-refs is-volatile-Write_{sb} sb = {}
      by (clarsimp simp add: Ghost_{sb} aargh)

    show ?thesis
    proof (cases a $\in$ R)

```
      case False
      with a-notin have a ∉ dom (𝒮 ⊕W R ⊖A L)
        by auto
      from Cons.hyps [OF out′ this share]
      show ?thesis
        by (auto simp add: Ghostsb)
    next
      case True
      with a-notin have augment-rels (dom 𝒮) R ℛ a = Some False
        by (auto simp add: augment-rels-def split: option.splits)
      from release-False-mono [of augment-rels (dom 𝒮) R ℛ, OF this out′]
      show ?thesis
        by (auto simp add: Ghostsb)
    qed
  next
    case Writesb with Cons show ?thesis by (clarsimp split: if-split-asm)
  next
    case Readsb with Cons show ?thesis by auto
  next
    case Progsb with Cons show ?thesis by auto
  qed
qed


lemma release-not-unshared-no-write:
  ⋀𝒮 ℛ. ⟦
     outstanding-refs is-volatile-Writesb sb = {};
  non-volatile-writes-unshared 𝒮 sb;
  release sb (dom 𝒮) ℛ a ≠  Some False;
  a ∈ dom (share sb 𝒮)⟧
   ⟹
    a ∉ outstanding-refs is-non-volatile-Writesb sb
proof (induct sb)
  case Nil thus ?case by (auto simp add: read-only-def)
next
  case (Cons x sb)
  have aargh: (Not ∘ is-volatile-Writesb) = (λa. ¬ is-volatile-Writesb a)
    by (rule ext) auto
  show ?case
  proof (cases x)
    case (Ghostsb A L R W)
    from Cons.prems obtain
      share: a ∈ dom (share sb (𝒮 ⊕W R ⊖A L)) and
      rel: release sb
           (dom (𝒮 ⊕W R ⊖A L)) (augment-rels (dom 𝒮) R ℛ) a ≠  Some False and
      out′: outstanding-refs is-volatile-Writesb sb = {} and
      nvu: non-volatile-writes-unshared (𝒮 ⊕W R ⊖A L) sb
      by (clarsimp simp add: Ghostsb )


    from Cons.hyps [OF out′ nvu rel share]
    show ?thesis by (auto simp add: Ghostsb)
  next
    case (Writesb volatile a′ sop v A L R W)
    show ?thesis
    proof (cases volatile)
      case True with Writesb Cons.prems have False by auto
      thus ?thesis ..
    next
      case False
      note not-vol = this
```

```
      from Cons.prems obtain
        rel: release sb (dom 𝒮) ℛ a ≠  Some False and
        out′: outstanding-refs is-volatile-Write_sb sb = {} and
        nvo: non-volatile-writes-unshared 𝒮 sb and
        a′-not-dom: a′ ∉ dom 𝒮 and
        a-dom: a ∈ dom (share sb 𝒮)
        by (auto simp add: Write_sb False)
      from Cons.hyps [OF out′ nvo rel a-dom]
      have a-notin-rest: a ∉ outstanding-refs is-non-volatile-Write_sb sb.

      show ?thesis
      proof (cases a′=a)
        case False with a-notin-rest
        show ?thesis by (clarsimp simp add: Write_sb  not-vol )
      next
        case True
        from shared-switch-release-False [OF out′ a′-not-dom [simplified True] a-dom]
        have release sb (dom 𝒮) ℛ a =  Some False.
        with rel have False by simp
        thus ?thesis ..
      qed
    qed
  next
    case Read_sb with Cons show ?thesis by auto
  next
    case Prog_sb with Cons show ?thesis by auto
  qed
qed

corollary release-not-unshared-no-write-take:
 assumes nvw: non-volatile-writes-unshared 𝒮 (takeWhile (Not ∘ is-volatile-Write_sb) sb)
 assumes rel: release (takeWhile (Not ∘ is-volatile-Write_sb) sb) (dom 𝒮) ℛ a ≠  Some False
 assumes a-in: a ∈ dom (share (takeWhile (Not ∘ is-volatile-Write_sb) sb) 𝒮)
 shows
    a ∉ outstanding-refs is-non-volatile-Write_sb (takeWhile (Not ∘ is-volatile-Write_sb) sb)
using release-not-unshared-no-write[OF takeWhile-not-vol-write-outstanding-refs [of sb] nvw rel a-in]
by simp


lemma read-only-unacquired-share′:
  ⋀S 𝒪. ⟦𝒪 ∩ read-only S = {}; weak-sharing-consistent 𝒪 sb; a ∈ read-only S;
  a ∉ all-shared sb; a ∉ acquired True sb 𝒪 ⟧
⟹ a ∈ read-only (share sb S)
proof (induct sb)
    case Nil thus ?case by simp
next
  case (Cons x sb)
  show ?case
  proof (cases x)
    case (Write_sb volatile a′ sop v A L R W)
    show ?thesis
    proof (cases volatile)
      case True
      note volatile=this
      from Cons.prems
      obtain a-ro: a ∈ read-only S and a-R: a ∉ R and a-unsh: a ∉ all-shared sb and
 owns-ro: 𝒪 ∩ read-only S = {} and
 L-A: L ⊆ A and A-R:  A ∩ R = {} and R-owns: R ⊆ 𝒪 and
 consis′: weak-sharing-consistent (𝒪 ∪ A − R) sb and
        a-notin: a ∉ acquired True sb (𝒪 ∪ A − R)
 by (clarsimp simp add: Write_sb True)
```

```
      show ?thesis
      proof (cases a ∈ A)
        case True
        with a-R have a ∈ 𝒪 ∪ A − R by auto
        from all-shared-acquired-in [OF this a-unsh]
        have a ∈  acquired True sb (𝒪 ∪ A − R) by auto
        with a-notin have False by auto
        thus ?thesis ..
      next
        case False

        from owns-ro A-R R-owns have owns-ro′: (𝒪 ∪ A − R) ∩ read-only (S ⊕W R ⊖A L) = {}
    by (auto simp add: in-read-only-convs)

        from a-ro False owns-ro R-owns L-A have a-ro′: a ∈ read-only (S ⊕W R ⊖A L)
    by (auto simp add: in-read-only-convs)
        from Cons.hyps [OF owns-ro′ consis′ a-ro′ a-unsh a-notin]
        show ?thesis
    by (clarsimp simp add: Write_sb True)
      qed
    next
      case False
      with Cons show ?thesis
  by (clarsimp simp add: Write_sb False)
    qed
  next
    case Read_sb with Cons show ?thesis by (clarsimp)
  next
    case Prog_sb with Cons show ?thesis by (clarsimp)
  next
    case (Ghost_sb A L R W)
    from Cons.prems
    obtain a-ro: a ∈ read-only S and a-R: a ∉ R and a-unsh: a ∉ all-shared sb and
      owns-ro: 𝒪 ∩ read-only S = {} and
      L-A: L ⊆ A and A-R:  A ∩ R = {} and R-owns: R ⊆ 𝒪 and
      consis′: weak-sharing-consistent (𝒪 ∪ A − R) sb and
      a-notin: a ∉ acquired True sb (𝒪 ∪ A − R)
      by (clarsimp simp add: Ghost_sb)
    show ?thesis
    proof (cases a ∈ A)
      case True
      with a-R have a ∈ 𝒪 ∪ A − R by auto
      from all-shared-acquired-in [OF this a-unsh]
      have a ∈  acquired True sb (𝒪 ∪ A − R) by auto
      with a-notin have False by auto
      thus ?thesis ..
    next
      case False

      from owns-ro A-R R-owns have owns-ro′: (𝒪 ∪ A − R) ∩ read-only (S ⊕W R ⊖A L) = {}
        by (auto simp add: in-read-only-convs)

      from a-ro False owns-ro R-owns L-A have a-ro′: a ∈ read-only (S ⊕W R ⊖A L)
        by (auto simp add: in-read-only-convs)
      from Cons.hyps [OF owns-ro′ consis′ a-ro′ a-unsh a-notin]
      show ?thesis
        by (clarsimp simp add: Ghost_sb)
    qed
  qed
qed
```

```
lemma read-only-share-all-until-volatile-write-unacquired':
  ⋀𝒮. ⟦ownership-distinct ts; read-only-unowned 𝒮 ts; weak-sharing-consis ts;
  ∀i < length ts. (let (-,-,-,sb,-,𝒪,ℛ) = ts!i in
        a ∉ acquired True (takeWhile (Not ∘ is-volatile-Write_sb) sb) 𝒪 ∧
        a ∉ all-shared (takeWhile (Not ∘ is-volatile-Write_sb) sb
    ));
  a ∈ read-only 𝒮⟧
  ⟹ a ∈ read-only (share-all-until-volatile-write ts 𝒮)
proof (induct ts)
  case Nil thus ?case by simp
next
  case (Cons t ts)
  obtain p is 𝒪 ℛ 𝒟 j sb where
    t: t = (p,is,j,sb,𝒟,𝒪,ℛ)
    by (cases t)

  have dist: ownership-distinct (t#ts) by fact
  then interpret ownership-distinct t#ts .
  from ownership-distinct-tl [OF dist]
  have dist': ownership-distinct ts.


  have aargh: (Not ∘ is-volatile-Write_sb) = (λa. ¬ is-volatile-Write_sb a)
    by (rule ext) auto

  have a-ro: a ∈ read-only 𝒮 by fact
  have ro-unowned: read-only-unowned 𝒮 (t#ts) by fact
  then interpret read-only-unowned 𝒮 t#ts .
  have consis: weak-sharing-consis (t#ts) by fact
  then interpret weak-sharing-consis t#ts .

  note consis' = weak-sharing-consis-tl [OF consis]

  let ?take-sb = (takeWhile (Not ∘ is-volatile-Write_sb) sb)
  let ?drop-sb = (dropWhile (Not ∘ is-volatile-Write_sb) sb)

  from weak-sharing-consis [of 0] t
  have consis-sb: weak-sharing-consistent 𝒪 sb
    by force
  with weak-sharing-consistent-append [of 𝒪 ?take-sb ?drop-sb]
  have consis-take: weak-sharing-consistent 𝒪 ?take-sb
    by auto


  have ro-unowned': read-only-unowned (share ?take-sb 𝒮) ts
  proof
    fix j
    fix p_j is_j 𝒪_j ℛ_j 𝒟_j j_j sb_j
    assume j-bound: j < length ts
    assume jth: ts!j = (p_j,is_j,j_j,sb_j,𝒟_j,𝒪_j,ℛ_j)
    show 𝒪_j ∩ read-only (share ?take-sb 𝒮) = {}
    proof -
      {
        fix a
        assume a-owns: a ∈ 𝒪_j
        assume a-ro: a ∈ read-only (share ?take-sb 𝒮)
        have False
        proof -
          from ownership-distinct [of 0 Suc j] j-bound jth t
          have dist: (𝒪 ∪ all-acquired sb) ∩ (𝒪_j ∪ all-acquired sb_j) = {}
            by fastforce
```

```
        from read-only-unowned [of Suc j] j-bound jth
        have dist-ro: 𝒪ⱼ ∩ read-only 𝒮 = {} by force
        show ?thesis
        proof (cases a ∈ (𝒪 ∪ all-acquired sb))
          case True
          with dist a-owns show False by auto
        next
          case False
          hence a ∉ (𝒪 ∪ all-acquired ?take-sb)
          using all-acquired-append [of ?take-sb ?drop-sb]
            by auto
          from read-only-share-unowned [OF consis-take this a-ro]
          have a ∈ read-only 𝒮.
          with dist-ro a-owns show False by auto
       qed
     qed
    }
    thus ?thesis by auto
  qed
qed


from Cons.prems
obtain unacq-ts: ∀i < length ts. (let (-,-,-,sb,-,𝒪,-) = ts!i in
         a ∉ acquired True (takeWhile (Not ∘ is-volatile-Write_sb) sb) 𝒪 ∧
          a ∉ all-shared (takeWhile (Not ∘ is-volatile-Write_sb) sb)) and
  unacq-sb: a ∉ acquired True (takeWhile (Not ∘ is-volatile-Write_sb) sb) 𝒪 and
  unsh-sb: a ∉ all-shared (takeWhile (Not ∘ is-volatile-Write_sb) sb)

 apply clarsimp
 apply (rule that)
 apply (auto simp add: t aargh)
 done


from read-only-unowned [of 0] t
have owns-ro: 𝒪 ∩ read-only 𝒮 = {}
  by force


from read-only-unacquired-share′ [OF owns-ro consis-take a-ro unsh-sb unacq-sb]
have a ∈ read-only (share (takeWhile (Not ∘ is-volatile-Write_sb) sb) 𝒮).
from Cons.hyps [OF dist′ ro-unowned′ consis′ unacq-ts this]
show ?case
  by (simp add: t)
qed




lemma not-shared-not-acquired-switch:
  ⋀X Y. ⟦a ∉ all-shared sb; a ∉ X; a ∉ acquired True sb X; a ∉ Y⟧ ⟹ a ∉ acquired True sb Y
proof (induct sb)
  case Nil thus ?case by simp
next
  case (Cons x sb)
  show ?case
  proof (cases x)
    case (Write_sb volatile a′ sop v A L R W)
    show ?thesis
```

proof (cases volatile)
case True
from Cons.prems **obtain**
a-X: a ∉ X **and** a-acq: a ∉ acquired True sb (X ∪ A − R) **and**
a-Y: a ∉ Y **and** a-R: a ∉ R **and**
a-shared: a ∉ all-shared sb
by (clarsimp simp add: Write_{sb} True)
show ?thesis
proof (cases a ∈ A)
case True
with a-X a-R
have a ∈ X ∪ A − R **by** auto
from all-shared-acquired-in [OF this a-shared]
have a ∈ acquired True sb (X ∪ A − R)**.**
with a-acq **have** False **by** simp
thus ?thesis **..**
next
case False
with a-X a-Y **obtain** a-X′: a ∉ X ∪ A − R **and** a-Y′: a ∉ Y ∪ A − R
by auto
from Cons.hyps [OF a-shared a-X′ a-acq a-Y′]
show ?thesis
by (auto simp add: Write_{sb} True)
qed
next
case False **with** Cons.hyps [of X Y] Cons.prems **show** ?thesis **by** (auto simp add: Write_{sb})
qed
next
case Read_{sb} **with** Cons.hyps [of X Y] Cons.prems **show** ?thesis **by** (auto)
next
case Prog_{sb} **with** Cons.hyps [of X Y] Cons.prems **show** ?thesis **by** (auto)
next
case (Ghost_{sb} A L R W)
from Cons.prems **obtain**
a-X: a ∉ X **and** a-acq: a ∉ acquired True sb (X ∪ A − R) **and**
a-Y: a ∉ Y **and** a-R: a ∉ R **and**
a-shared: a ∉ all-shared sb
by (clarsimp simp add: Ghost_{sb})
show ?thesis
proof (cases a ∈ A)
case True
with a-X a-R
have a ∈ X ∪ A − R **by** auto
from all-shared-acquired-in [OF this a-shared]
have a ∈ acquired True sb (X ∪ A − R)**.**
with a-acq **have** False **by** simp
thus ?thesis **..**
next
case False
with a-X a-Y **obtain** a-X′: a ∉ X ∪ A − R **and** a-Y′: a ∉ Y ∪ A − R
by auto
from Cons.hyps [OF a-shared a-X′ a-acq a-Y′]
show ?thesis
by (auto simp add: Ghost_{sb})
qed
qed
qed

lemma read-only-share-all-acquired-in′:
⋀S 𝒪. ⟦𝒪 ∩ read-only S = {}; weak-sharing-consistent 𝒪 sb; a ∈ read-only (share sb S)⟧

$\Longrightarrow$ a $\in$ read-only (share sb Map.empty) $\vee$ (a $\in$ read-only S $\wedge$ a $\notin$ acquired True sb $\mathcal{O}$ $\wedge$ a $\notin$ all-shared sb)
proof (induct sb)
  case Nil **thus** ?case **by** auto
next
  case (Cons x sb)
  show ?case
  proof (cases x)
    case ($Write_{sb}$ volatile a′ sop v A L R W)
    show ?thesis
    proof (cases volatile)
      case True
      note volatile=this
      from Cons.prems
      obtain a-in: a $\in$ read-only (share sb (S $\oplus_W$ R $\ominus_A$ L)) **and**
owns-ro: $\mathcal{O}$ $\cap$ read-only S = {} **and**
L-A: L $\subseteq$ A **and** A-R: A $\cap$ R = {} **and** R-owns: R $\subseteq$ $\mathcal{O}$ **and**
consis′: weak-sharing-consistent ($\mathcal{O}$ $\cup$ A − R) sb
by (clarsimp simp add: $Write_{sb}$ True)

      from owns-ro A-R R-owns **have** owns-ro′: ($\mathcal{O}$ $\cup$ A − R) $\cap$ read-only (S $\oplus_W$ R $\ominus_A$ L) = {}
by (auto simp add: in-read-only-convs)

      from Cons.hyps [OF owns-ro′ consis′ a-in]
      have hyp: a $\in$ read-only (share sb Map.empty) $\vee$
            (a $\in$ read-only (S $\oplus_W$ R $\ominus_A$ L) $\wedge$ a $\notin$ acquired True sb ($\mathcal{O}$ $\cup$ A − R) $\wedge$ a $\notin$ all-shared sb).

      have a $\in$ read-only (share sb (Map.empty $\oplus_W$ R $\ominus_A$ L)) $\vee$
        (a $\in$ read-only S $\wedge$ a $\notin$ R $\wedge$ a $\notin$ acquired True sb ($\mathcal{O}$ $\cup$ A − R) $\wedge$ a $\notin$ all-shared sb)
      proof −
{
  assume a-emp: a $\in$ read-only (share sb Map.empty)
  have read-only Map.empty $\subseteq$ read-only (Map.empty $\oplus_W$ R $\ominus_A$ L)
    by (auto simp add: in-read-only-convs)

  from share-read-only-mono-in [OF a-emp this]
  have a $\in$ read-only (share sb (Map.empty $\oplus_W$ R $\ominus_A$ L)).
}
moreover
{
  assume a-ro: a $\in$ read-only (S $\oplus_W$ R $\ominus_A$ L) **and**
      a-not-acq: a $\notin$ acquired True sb ($\mathcal{O}$ $\cup$ A − R) **and**
      a-unsh: a $\notin$ all-shared sb
    have ?thesis
  proof (cases a $\in$ read-only S)
    case True
    with a-ro **obtain** a-A: a $\notin$ A
      by (auto simp add: in-read-only-convs)
      with True a-not-acq a-unsh R-owns owns-ro
      show ?thesis
        by auto
    next
      case False
    with a-ro **have** a-ro-empty: a $\in$ read-only (Map.empty $\oplus_W$ R $\ominus_A$ L)
      by (auto simp add: in-read-only-convs split: if-split-asm)

    have read-only (Map.empty $\oplus_W$ R $\ominus_A$ L) $\subseteq$ read-only (S $\oplus_W$ R $\ominus_A$ L)
      by (auto simp add: in-read-only-convs)
    with owns-ro′
    have owns-ro-empty: ($\mathcal{O}$ $\cup$ A − R) $\cap$ read-only (Map.empty $\oplus_W$ R $\ominus_A$ L) = {}
      by blast

```
    from read-only-unacquired-share′ [OF owns-ro-empty consis′ a-ro-empty a-unsh a-not-acq]
    have a ∈ read-only (share sb (Map.empty ⊕W R ⊖A L)).
    thus ?thesis
      by simp
  qed
}
moreover note hyp
ultimately show ?thesis by blast
     qed

     then show ?thesis
by (clarsimp simp add: Writesb True)
   next
     case False with Cons show ?thesis
by (auto simp add: Writesb)
   qed
 next
   case Readsb with Cons show ?thesis by auto
 next
   case Progsb with Cons show ?thesis by auto
 next
   case (Ghostsb A L R W)
   from Cons.prems
   obtain a-in: a ∈ read-only (share sb (S ⊕W R ⊖A L)) and
     owns-ro: 𝒪 ∩ read-only S = {} and
     L-A: L ⊆ A and A-R:  A ∩ R = {} and R-owns: R ⊆ 𝒪 and
     consis′: weak-sharing-consistent (𝒪 ∪ A − R) sb
     by (clarsimp simp add: Ghostsb)

   from owns-ro A-R R-owns have owns-ro′: (𝒪 ∪ A − R) ∩ read-only (S ⊕W R ⊖A L) = {}
     by (auto simp add: in-read-only-convs)

   from Cons.hyps [OF owns-ro′ consis′ a-in]
   have hyp: a ∈ read-only (share sb Map.empty) ∨
             (a ∈ read-only (S ⊕W R ⊖A L) ∧ a ∉ acquired True sb (𝒪 ∪ A − R) ∧ a ∉ all-shared sb).

   have a ∈ read-only (share sb (Map.empty ⊕W R ⊖A L)) ∨
         (a ∈ read-only S ∧ a ∉ R ∧ a ∉ acquired True sb (𝒪 ∪ A − R) ∧ a ∉ all-shared sb)
   proof −
    {
assume a-emp: a ∈ read-only (share sb Map.empty)
have read-only Map.empty ⊆ read-only (Map.empty ⊕W R ⊖A L)
  by (auto simp add: in-read-only-convs)

from share-read-only-mono-in [OF a-emp this]
have a ∈ read-only (share sb (Map.empty ⊕W R ⊖A L)).
     }
     moreover
     {
assume a-ro: a ∈ read-only (S ⊕W R ⊖A L) and
        a-not-acq: a ∉ acquired True sb (𝒪 ∪ A − R) and
        a-unsh: a ∉ all-shared sb
      have ?thesis
      proof (cases a ∈ read-only S)
  case True
  with a-ro obtain a-A: a ∉ A
    by (auto simp add: in-read-only-convs)
        with True a-not-acq a-unsh R-owns owns-ro
        show ?thesis
          by auto
```

```
        next
          case False
   with a-ro have a-ro-empty: a ∈ read-only (Map.empty ⊕W R ⊖A L)
     by (auto simp add: in-read-only-convs split: if-split-asm)

   have read-only (Map.empty ⊕W R ⊖A L) ⊆ read-only (S ⊕W R ⊖A L)
     by (auto simp add: in-read-only-convs)
   with owns-ro'
   have owns-ro-empty: (𝒪 ∪ A − R) ∩ read-only (Map.empty ⊕W R ⊖A L) = {}
     by blast


   from read-only-unacquired-share' [OF owns-ro-empty consis' a-ro-empty a-unsh a-not-acq]
   have a ∈ read-only (share sb (Map.empty ⊕W R ⊖A L)).
   thus ?thesis
     by simp
 qed
     }
     moreover note hyp
     ultimately show ?thesis by blast
    qed

    then show ?thesis
      by (clarsimp simp add: Ghost_sb)
  qed
qed

lemma in-read-only-share-all-until-volatile-write':
  assumes dist: ownership-distinct ts
  assumes consis: sharing-consis 𝒮 ts
  assumes ro-unowned: read-only-unowned 𝒮 ts
  assumes i-bound: i < length ts
  assumes ts-i: ts!i = (p,is,j,sb,𝒟,𝒪,ℛ)
  assumes a-unacquired-others: ∀j < length ts. i≠j ⟶
          (let (-,-,-,sb_j,-,𝒪,-) = ts!j in
          a ∉ acquired True (takeWhile (Not ∘ is-volatile-Write_sb) sb_j) 𝒪 ∧
          a ∉ all-shared (takeWhile (Not ∘ is-volatile-Write_sb) sb_j ))
  assumes a-ro-share: a ∈ read-only (share sb 𝒮)
  shows a ∈ read-only (share (dropWhile (Not ∘ is-volatile-Write_sb) sb)
                 (share-all-until-volatile-write ts 𝒮))
proof −
  from consis
  interpret sharing-consis 𝒮 ts .
  interpret read-only-unowned 𝒮 ts by fact

  from sharing-consis [OF i-bound ts-i]
  have consis-sb: sharing-consistent 𝒮 𝒪 sb.
  from sharing-consistent-weak-sharing-consistent [OF this]
  have weak-consis: weak-sharing-consistent 𝒪 sb.
  from read-only-unowned [OF i-bound ts-i]
  have owns-ro: 𝒪 ∩ read-only 𝒮 = {}.
  from read-only-share-all-acquired-in' [OF owns-ro weak-consis a-ro-share]

  have a ∈ read-only (share sb Map.empty) ∨ a ∈ read-only 𝒮 ∧ a ∉ acquired True sb 𝒪 ∧ a ∉ all-shared sb.
  moreover

  let ?take-sb = (takeWhile (Not ∘ is-volatile-Write_sb) sb)
  let ?drop-sb = (dropWhile (Not ∘ is-volatile-Write_sb) sb)

  from weak-consis weak-sharing-consistent-append [of 𝒪 ?take-sb ?drop-sb]
  obtain weak-consis': weak-sharing-consistent (acquired True ?take-sb 𝒪) ?drop-sb and
```

weak-consis-take: weak-sharing-consistent $\mathcal{O}$?take-sb
by auto

{
assume a $\in$ read-only (share sb Map.empty)
with share-append [of ?take-sb ?drop-sb]
have a-in$'$: a $\in$ read-only (share ?drop-sb (share ?take-sb Map.empty))
by auto

have owns-empty: $\mathcal{O}$ $\cap$ read-only Map.empty = {}
by auto

from weak-sharing-consistent-preserves-distinct [OF weak-consis-take owns-empty]
have acquired True ?take-sb $\mathcal{O}$ $\cap$ read-only (share ?take-sb Map.empty) = {}**.**

from read-only-share-all-acquired-in [OF this weak-consis$'$ a-in$'$]
have a $\in$ read-only (share ?drop-sb Map.empty) $\vee$ a $\in$ read-only (share ?take-sb Map.empty) $\wedge$ a $\notin$ all-acquired ?drop-sb**.**
moreover
{
assume a-ro-drop: a $\in$ read-only (share ?drop-sb Map.empty)
have read-only Map.empty $\subseteq$ read-only (share-all-until-volatile-write ts $\mathcal{S}$)
by auto
from share-read-only-mono-in [OF a-ro-drop this]
have ?thesis **.**
}
moreover
{
assume a-ro-take: a $\in$ read-only (share ?take-sb Map.empty)
assume a-unacq-drop: a $\notin$ all-acquired ?drop-sb
from read-only-share-unowned-in [OF weak-consis-take a-ro-take]
have a $\in$ $\mathcal{O}$ $\cup$ all-acquired ?take-sb **by** auto
hence a $\in$ $\mathcal{O}$ $\cup$ all-acquired sb **using** all-acquired-append [of ?take-sb ?drop-sb]
by auto
from share-all-until-volatile-write-thread-local$'$ [OF dist consis i-bound ts-i this] a-ro-share
have ?thesis **by** (auto simp add: read-only-def)
}
ultimately have ?thesis **by** blast
}

moreover

{
assume a-ro: a $\in$ read-only $\mathcal{S}$
assume a-unacq: a $\notin$ acquired True sb $\mathcal{O}$
assume a-unsh: a $\notin$ all-shared sb
with all-shared-append [of ?take-sb ?drop-sb]
obtain a-notin-take: a $\notin$ all-shared ?take-sb **and** a-notin-drop: a $\notin$ all-shared ?drop-sb
by auto
have ?thesis
proof (cases a $\in$ acquired True ?take-sb $\mathcal{O}$)
case True
from all-shared-acquired-in [OF this a-notin-drop] acquired-append [of True ?take-sb ?drop-sb $\mathcal{O}$] a-unacq
have False
by auto
thus ?thesis **..**
next
case False
with a-unacquired-others i-bound ts-i a-notin-take
have a-unacq$'$: $\forall$j < length ts.
(let (-,-,-,sb_j,-,$\mathcal{O}$,-) = ts!j in

```
        a ∉ acquired True (takeWhile (Not ∘ is-volatile-Write_sb) sb_j) 𝒪 ∧
        a ∉ all-shared (takeWhile (Not ∘ is-volatile-Write_sb) sb_j ))
      by (auto simp add: Let-def)

    from local.weak-sharing-consis-axioms have weak-sharing-consis ts .
    from read-only-share-all-until-volatile-write-unacquired' [OF dist ro-unowned
     ‹weak-sharing-consis ts› a-unacq' a-ro]
    have a-ro-all: a ∈ read-only (share-all-until-volatile-write ts 𝒮) .

    from weak-consis weak-sharing-consistent-append [of 𝒪 ?take-sb ?drop-sb]
    have weak-consis-drop: weak-sharing-consistent (acquired True ?take-sb 𝒪) ?drop-sb
      by auto

    from weak-sharing-consistent-preserves-distinct-share-all-until-volatile-write [OF dist
      ro-unowned ‹weak-sharing-consis ts› i-bound ts-i]
    have acquired True ?take-sb 𝒪 ∩
       read-only (share-all-until-volatile-write ts 𝒮) = {}.

    from read-only-unacquired-share' [OF this weak-consis-drop a-ro-all a-notin-drop]
      acquired-append [of True ?take-sb ?drop-sb 𝒪] a-unacq
    show ?thesis by auto
  qed
 }
 ultimately show ?thesis by blast
qed

lemma all-acquired-unshared-acquired:
 ⋀𝒪. a ∈ all-acquired sb ==> a ∉ all-shared sb ==> a ∈ acquired True sb 𝒪
apply (induct sb)
apply (auto split: memref.split intro: all-shared-acquired-in)
done




lemma  safe-RMW-common:
 assumes safe: 𝒪s,ℛs,i⊢ (RMW a t (D,f) cond ret A L R W# is, j, m, 𝒟, 𝒪, 𝒮)√
 shows (a ∈ 𝒪 ∨ a ∈ dom 𝒮) ∧ (∀ j < length 𝒪s. i≠j ⟶ (ℛs!j) a ≠ Some False)
using safe
apply (cases)
apply (auto simp add: domIff)
done


lemma acquired-reads-all-acquired': ⋀𝒪.
 acquired-reads True sb 𝒪 ⊆ acquired True sb 𝒪 ∪ all-shared sb
apply (induct sb)
apply  clarsimp
apply (auto split: memref.splits dest: all-shared-acquired-in)
done


lemma release-all-shared-exchange:
 ⋀ℛ S' S. ∀ a ∈ all-shared sb. (a ∈ S') = (a ∈ S) ⟹ release sb S' ℛ = release sb S ℛ
proof (induct sb)
 case Nil thus ?case by auto
next
 case (Cons x sb)
 show ?case
 proof (cases x)
  case (Write_sb volatile a' sop v A L R W)
```

show ?thesis
proof (cases volatile)
case True
note volatile=this
from Cons.hyps [of ($S' \cup R - L$) ($S \cup R - L$) Map.empty] Cons.prems
show ?thesis
by (auto simp add: $Write_{sb}$ volatile)
next
case False with Cons $Write_{sb}$ show ?thesis by auto
qed
next
case $Read_{sb}$ with Cons show ?thesis by auto
next
case $Prog_{sb}$ with Cons show ?thesis by auto
next
case ($Ghost_{sb}$ A L R W)
from augment-rels-shared-exchange [of R S S' $\mathcal{R}$] Cons.prems
have augment-rels S' R $\mathcal{R}$ = augment-rels S R $\mathcal{R}$
by (auto simp add: $Ghost_{sb}$)

with Cons.hyps [of ($S' \cup R - L$) ($S \cup R - L$) augment-rels S R $\mathcal{R}$] Cons.prems
show ?thesis
by (auto simp add: $Ghost_{sb}$)
qed
qed

lemma release-append-$Prog_{sb}$:
$\bigwedge$S $\mathcal{R}$. (release (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) (sb @ [$Prog_{sb}$ p_1 p_2 mis])) S $\mathcal{R}$) =
(release (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb) S $\mathcal{R}$)
by (induct sb) (auto split: memref.splits)

A.5 Simulation of Store Buffer Machine with History by Virtual Machine with Delayed Releases

theorem (**in** xvalid-program) concurrent-direct-steps-simulates-store-buffer-history-step:
assumes step-sb: $(ts_{sb},m_{sb},\mathcal{S}_{sb}) \Rightarrow_{sbh} (ts_{sb}',m_{sb}',\mathcal{S}_{sb}')$
assumes valid-own: valid-ownership $\mathcal{S}_{sb}$ ts_{sb}
assumes valid-sb-reads: valid-reads m_{sb} ts_{sb}
assumes valid-hist: valid-history program-step ts_{sb}
assumes valid-sharing: valid-sharing $\mathcal{S}_{sb}$ ts_{sb}
assumes tmps-distinct: tmps-distinct ts_{sb}
assumes valid-sops: valid-sops ts_{sb}
assumes valid-dd: valid-data-dependency ts_{sb}
assumes load-tmps-fresh: load-tmps-fresh ts_{sb}
assumes enough-flushs: enough-flushs ts_{sb}
assumes valid-program-history: valid-program-history ts_{sb}
assumes valid: valid ts_{sb}
assumes sim: $(ts_{sb},m_{sb},\mathcal{S}_{sb}) \sim (ts,m,\mathcal{S})$
assumes safe-reach: safe-reach-direct safe-delayed $(ts,m,\mathcal{S})$
shows valid-ownership $\mathcal{S}_{sb}'$ ts_{sb}' $\wedge$ valid-reads m_{sb}' ts_{sb}' $\wedge$ valid-history program-step ts_{sb}' $\wedge$
valid-sharing $\mathcal{S}_{sb}'$ ts_{sb}' $\wedge$ tmps-distinct ts_{sb}' $\wedge$ valid-data-dependency ts_{sb}' $\wedge$
valid-sops ts_{sb}' $\wedge$ load-tmps-fresh ts_{sb}' $\wedge$ enough-flushs ts_{sb}' $\wedge$
valid-program-history ts_{sb}' $\wedge$ valid ts_{sb}' $\wedge$
$(\exists ts'\ \mathcal{S}'\ m'.\ (ts,m,\mathcal{S}) \Rightarrow_d^* (ts',m',\mathcal{S}') \wedge$

$(\mathrm{ts_{sb}}',\mathrm{m_{sb}}',\mathcal{S}_{\mathsf{sb}}') \sim (\mathrm{ts}',\mathrm{m}',\mathcal{S}'))$

proof −

interpret direct-computation:
computation direct-memop-step empty-storebuffer-step program-step λp p′ is sb. sb **.**
interpret sbh-computation:
computation sbh-memop-step flush-step program-step
λp p′ is sb. sb @ [$\mathrm{Prog_{sb}}$ p p′ is] **.**
interpret valid-ownership $\mathcal{S}_{\mathsf{sb}}$ $\mathrm{ts_{sb}}$ **by** fact
interpret valid-reads $\mathrm{m_{sb}}$ $\mathrm{ts_{sb}}$ **by** fact
interpret valid-history program-step $\mathrm{ts_{sb}}$ **by** fact
interpret valid-sharing $\mathcal{S}_{\mathsf{sb}}$ $\mathrm{ts_{sb}}$ **by** fact
interpret tmps-distinct $\mathrm{ts_{sb}}$ **by** fact
interpret valid-sops $\mathrm{ts_{sb}}$ **by** fact
interpret valid-data-dependency $\mathrm{ts_{sb}}$ **by** fact
interpret load-tmps-fresh $\mathrm{ts_{sb}}$ **by** fact
interpret enough-flushs $\mathrm{ts_{sb}}$ **by** fact
interpret valid-program-history $\mathrm{ts_{sb}}$ **by** fact
from valid-own valid-sharing
have valid-own-sharing: valid-ownership-and-sharing $\mathcal{S}_{\mathsf{sb}}$ $\mathrm{ts_{sb}}$
by (simp add: valid-sharing-def valid-ownership-and-sharing-def)
then
interpret valid-ownership-and-sharing $\mathcal{S}_{\mathsf{sb}}$ $\mathrm{ts_{sb}}$ **.**

from safe-reach-safe-refl [OF safe-reach]
have safe: safe-delayed (ts,m,$\mathcal{S}$)**.**

from step-sb
show ?thesis
proof (cases)
case (Memop i $\mathrm{p_{sb}}$ $\mathrm{is_{sb}}$ $\mathrm{j_{sb}}$ sb $\mathcal{D}_{\mathsf{sb}}$ $\mathcal{O}_{\mathsf{sb}}$ $\mathcal{R}_{\mathsf{sb}}$ $\mathrm{is_{sb}}'$ $\mathrm{j_{sb}}'$ sb′ $\mathcal{D}_{\mathsf{sb}}'$ $\mathcal{O}_{\mathsf{sb}}'$ $\mathcal{R}_{\mathsf{sb}}'$)
then obtain
$\mathrm{ts_{sb}}'$: $\mathrm{ts_{sb}}' = \mathrm{ts_{sb}}[i := (\mathrm{p_{sb}}, \mathrm{is_{sb}}',\mathrm{j_{sb}}', \mathrm{sb}', \mathcal{D}_{\mathsf{sb}}', \mathcal{O}_{\mathsf{sb}}',\mathcal{R}_{\mathsf{sb}}')]$ **and**
i-bound: i < length $\mathrm{ts_{sb}}$ **and**
$\mathrm{ts_{sb}}$-i: $\mathrm{ts_{sb}}$! i = ($\mathrm{p_{sb}}$, $\mathrm{is_{sb}}$,$\mathrm{j_{sb}}$,sb, $\mathcal{D}_{\mathsf{sb}}$, $\mathcal{O}_{\mathsf{sb}}$,$\mathcal{R}_{\mathsf{sb}}$) **and**
sbh-step: ($\mathrm{is_{sb}}$, $\mathrm{j_{sb}}$, sb, $\mathrm{m_{sb}}$, $\mathcal{D}_{\mathsf{sb}}$, $\mathcal{O}_{\mathsf{sb}}$, $\mathcal{R}_{\mathsf{sb}}$,$\mathcal{S}_{\mathsf{sb}}$) $\rightarrow_{\mathsf{sbh}}$
($\mathrm{is_{sb}}'$, $\mathrm{j_{sb}}'$, sb′, $\mathrm{m_{sb}}'$, $\mathcal{D}_{\mathsf{sb}}'$, $\mathcal{O}_{\mathsf{sb}}'$, $\mathcal{R}_{\mathsf{sb}}'$, $\mathcal{S}_{\mathsf{sb}}'$)
by auto

from sim **obtain**
m: m = flush-all-until-volatile-write $\mathrm{ts_{sb}}$ $\mathrm{m_{sb}}$ **and**
$\mathcal{S}$: $\mathcal{S}$ = share-all-until-volatile-write $\mathrm{ts_{sb}}$ $\mathcal{S}_{\mathsf{sb}}$ **and**
leq: length $\mathrm{ts_{sb}}$ = length ts **and**
ts-sim: $\forall$i<length $\mathrm{ts_{sb}}$.
let (p, $\mathrm{is_{sb}}$, j, sb, $\mathcal{D}_{\mathsf{sb}}$, $\mathcal{O}_{\mathsf{sb}}$,$\mathcal{R}$) = $\mathrm{ts_{sb}}$! i;
suspends = dropWhile (Not $\circ$ is-volatile-$\mathrm{Write_{sb}}$) sb
in $\exists$is $\mathcal{D}$. instrs suspends @ $\mathrm{is_{sb}}$ = is @ prog-instrs suspends $\wedge$
$\mathcal{D}_{\mathsf{sb}}$ = ($\mathcal{D}$ $\vee$ outstanding-refs is-volatile-$\mathrm{Write_{sb}}$ sb $\neq$ {}) $\wedge$
ts ! i =

(hd-prog p suspends,
is,
j |' (dom j − read-tmps suspends), (),
$\mathcal{D}$,
acquired True (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb) $\mathcal{O}_{\mathsf{sb}}$,
release (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb) (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}$)
by cases blast

from i-bound leq **have** i-bound′: i < length ts
by auto

have split-sb: sb = takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb @ dropWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb
(**is** sb = ?take-sb@?drop-sb)
by simp

from ts-sim [rule-format, OF i-bound] ts$_{\mathsf{sb}}$-i **obtain** suspends is $\mathcal{D}$ **where**
suspends: suspends = dropWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb **and**
is-sim: instrs suspends @ is$_{\mathsf{sb}}$ = is @ prog-instrs suspends **and**
$\mathcal{D}$: $\mathcal{D}_{\mathsf{sb}}$ = ($\mathcal{D}$ ∨ outstanding-refs is-volatile-Write$_{\mathsf{sb}}$ sb ≠ {}) **and**
ts-i: ts ! i =
(hd-prog p$_{\mathsf{sb}}$ suspends, is,
j$_{\mathsf{sb}}$ |' (dom j$_{\mathsf{sb}}$ − read-tmps suspends), (), $\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{\mathsf{sb}}$,
release ?take-sb (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}_{\mathsf{sb}}$)
by (auto simp add: Let-def)

from sbh-step-preserves-valid [OF i-bound ts$_{\mathsf{sb}}$-i sbh-step valid]
have valid′: valid ts$_{\mathsf{sb}}$′
by (simp add: ts$_{\mathsf{sb}}$′)

from $\mathcal{D}$ **have** $\mathcal{D}_{\mathsf{sb}}$: $\mathcal{D}_{\mathsf{sb}}$ = ($\mathcal{D}$ ∨ outstanding-refs is-volatile-Write$_{\mathsf{sb}}$?drop-sb ≠ {})
apply −
apply (case-tac outstanding-refs is-volatile-Write$_{\mathsf{sb}}$ sb = {})
apply (fastforce simp add: outstanding-refs-conv dest: set-dropWhileD)
apply (clarsimp)
apply (drule outstanding-refs-non-empty-dropWhile)
apply blast
done

let ?ts′ = ts[i := (p$_{\mathsf{sb}}$, is$_{\mathsf{sb}}$, j$_{\mathsf{sb}}$, (), $\mathcal{D}_{\mathsf{sb}}$, acquired True sb $\mathcal{O}_{\mathsf{sb}}$,
release sb (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}_{\mathsf{sb}}$)]
have i-bound-ts′: i < length ?ts′
using i-bound′
by auto
hence ts′-i: ?ts′!i = (p$_{\mathsf{sb}}$, is$_{\mathsf{sb}}$, j$_{\mathsf{sb}}$, (),
$\mathcal{D}_{\mathsf{sb}}$, acquired True sb $\mathcal{O}_{\mathsf{sb}}$, release sb (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}_{\mathsf{sb}}$)
by simp

from local.sharing-consis-axioms

have sharing-consis-ts_{sb}: sharing-consis $\mathcal{S}_{sb}$ ts_{sb} **.**
from sharing-consis [OF i-bound ts_{sb}-i]
have sharing-consis-sb: sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ sb**.**
from sharing-consistent-weak-sharing-consistent [OF this]
have weak-consis-sb: weak-sharing-consistent $\mathcal{O}_{sb}$ sb**.**
from this weak-sharing-consistent-append [of $\mathcal{O}_{sb}$?take-sb ?drop-sb]
have weak-consis-drop:weak-sharing-consistent (acquired True ?take-sb $\mathcal{O}_{sb}$) ?drop-sb
by auto
from local.ownership-distinct-axioms
have ownership-distinct-ts_{sb}: ownership-distinct ts_{sb} **.**
have steps-flush-sb: (ts,m,$\mathcal{S}$) $\Rightarrow_{d}^{*}$ (?ts$'$, flush ?drop-sb m, share ?drop-sb $\mathcal{S}$)
proof –
from valid-reads [OF i-bound ts_{sb}-i]
have reads-consis: reads-consistent False $\mathcal{O}_{sb}$ m_{sb} sb**.**
from reads-consistent-drop-volatile-writes-no-volatile-reads [OF this]
have no-vol-read: outstanding-refs is-volatile-$Read_{sb}$?drop-sb = {}**.**
from valid-program-history [OF i-bound ts_{sb}-i]
have causal-program-history is_{sb} sb**.**
then have cph: causal-program-history is_{sb} ?drop-sb
apply –
apply (rule causal-program-history-suffix [**where** sb=?take-sb])
apply (simp)
done
from valid-last-prog [OF i-bound ts_{sb}-i] **have** last-prog: last-prog p_{sb} sb = p_{sb}**.**
then
have lp: last-prog p_{sb} ?drop-sb = p_{sb}
apply –
apply (rule last-prog-same-append [**where** sb=?take-sb])
apply simp
done

from reads-consistent-flush-all-until-volatile-write [OF valid-own-sharing i-bound ts_{sb}-i reads-consis]
have reads-consis-m: reads-consistent True (acquired True ?take-sb $\mathcal{O}_{sb}$) m ?drop-sb
by (simp add: m)

from valid-history [OF i-bound ts_{sb}-i]
have h-consis: history-consistent j_{sb} (hd-prog p_{sb} (?take-sb@?drop-sb)) (?take-sb@?drop-sb)
by (simp)

have last-prog-hd-prog: last-prog (hd-prog p_{sb} sb) ?take-sb = (hd-prog p_{sb} ?drop-sb)
proof –
from last-prog-hd-prog-append$'$ [OF h-consis] last-prog
have last-prog (hd-prog p_{sb} ?drop-sb) ?take-sb = hd-prog p_{sb} ?drop-sb
by (simp)
moreover
have last-prog (hd-prog p_{sb} (?take-sb @ ?drop-sb)) ?take-sb =
last-prog (hd-prog p_{sb} ?drop-sb) ?take-sb
by (rule last-prog-hd-prog-append)

ultimately show ?thesis
by (simp)
qed

from valid-write-sops [OF i-bound ts_{sb}-i]
have $\forall$sop$\in$write-sops (?take-sb@?drop-sb). valid-sop sop
by (simp)
then obtain valid-sops-take: $\forall$sop$\in$write-sops ?take-sb. valid-sop sop **and**
valid-sops-drop: $\forall$sop$\in$write-sops ?drop-sb. valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF i-bound ts_{sb}-i]
have distinct-read-tmps (?take-sb@?drop-sb)
by (simp)
then obtain
read-tmps-take-drop: read-tmps ?take-sb $\cap$ read-tmps ?drop-sb = {} **and**
distinct-read-tmps-drop: distinct-read-tmps ?drop-sb
by (simp only: distinct-read-tmps-append)

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop h-consis]

have hist-consis$'$: history-consistent j_{sb} (hd-prog p_{sb} ?drop-sb) ?drop-sb
by (simp add: last-prog-hd-prog)

have rel-eq: release ?drop-sb (dom $\mathcal{S}$) (release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$) =
release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$
proof –
from release-append [of ?take-sb ?drop-sb]
have release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$ =
release ?drop-sb (dom (share ?take-sb $\mathcal{S}_{sb}$)) (release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
by simp
also
have dist: ownership-distinct ts_{sb} **by** fact
have consis: sharing-consis $\mathcal{S}_{sb}$ ts_{sb} **by** fact

have release ?drop-sb (dom (share ?take-sb $\mathcal{S}_{sb}$)) (release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
=
release ?drop-sb (dom $\mathcal{S}$) (release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
apply (simp only: $\mathcal{S}$)
apply (rule release-shared-exchange-weak [rule-format, OF - weak-consis-drop])
apply (rule share-all-until-volatile-write-thread-local [OF dist consis i-bound ts_{sb}-i,
symmetric])
using acquired-all-acquired [of True ?take-sb $\mathcal{O}_{sb}$] all-acquired-append [of ?take-sb
?drop-sb]
by auto
finally
show ?thesis **by** simp
qed

from flush-store-buffer [OF i-bound′ is-sim [simplified suspends] cph ts-i [simplified suspends] refl lp reads-consis-m hist-consis′ valid-sops-drop distinct-read-tmps-drop no-volatile-Read$_{sb}$-volatile-reads-consistent [OF no-vol-read], of $\mathcal{S}$]

show ?thesis **by** (simp add: acquired-take-drop [**where** pending-write=True, simplified] $\mathcal{D}_{sb}$ rel-eq)

qed

from safe-reach-safe-rtrancl [OF safe-reach steps-flush-sb]
have safe-ts′: safe-delayed (?ts′, flush ?drop-sb m, share ?drop-sb $\mathcal{S}$).
from safe-delayedE [OF safe-ts′ i-bound-ts′ ts′-i]
have safe-memop-flush-sb: map owned ?ts′,map released ?ts′,i$\vdash$
(is$_{sb}$, j$_{sb}$, flush ?drop-sb m, $\mathcal{D}_{sb}$,acquired True sb $\mathcal{O}_{sb}$,
share ?drop-sb $\mathcal{S}$) $\surd$.

from acquired-takeWhile-non-volatile-Write$_{sb}$
have acquired-take-sb: acquired True ?take-sb $\mathcal{O}_{sb} \subseteq \mathcal{O}_{sb} \cup$ all-acquired ?take-sb .

from sbh-step
show ?thesis
proof (cases)
case (SBHReadBuffered a v volatile t)
then obtain
is$_{sb}$: is$_{sb}$ = Read volatile a t # is$_{sb}$′ **and**
$\mathcal{O}_{sb}$′: $\mathcal{O}_{sb}$′=$\mathcal{O}_{sb}$ **and**
$\mathcal{D}_{sb}$′: $\mathcal{D}_{sb}$′=$\mathcal{D}_{sb}$ **and**
j$_{sb}$′: j$_{sb}$′ = j$_{sb}$(t$\mapsto$v) **and**
sb′: sb′=sb@[Read$_{sb}$ volatile a t v] **and**
m$_{sb}$′: m$_{sb}$′ = m$_{sb}$ **and**
$\mathcal{S}_{sb}$′: $\mathcal{S}_{sb}$′=$\mathcal{S}_{sb}$ **and**
$\mathcal{R}_{sb}$′: $\mathcal{R}_{sb}$′=$\mathcal{R}_{sb}$ **and**
buf-v: buffered-val sb a = Some v
by auto

from safe-memop-flush-sb [simplified is$_{sb}$]
obtain access-cond′: a $\in$ acquired True sb $\mathcal{O}_{sb}$ $\vee$
a $\in$ read-only (share ?drop-sb $\mathcal{S}$) $\vee$
(volatile $\wedge$ a $\in$ dom (share ?drop-sb $\mathcal{S}$)) **and**
volatile-clean: volatile $\longrightarrow \neg$ $\mathcal{D}_{sb}$ **and**
rels-cond: $\forall$ j < length ts. i$\neq$j $\longrightarrow$ released (ts!j) a $\neq$ Some False **and**
rels-nv-cond: $\neg$volatile $\longrightarrow$ ($\forall$ j < length ts. i$\neq$j $\longrightarrow$ a $\notin$ dom (released (ts!j)))
by cases auto

from clean-no-outstanding-volatile-Write$_{sb}$ [OF i-bound ts$_{sb}$-i] volatile-clean
have volatile-cond: volatile $\longrightarrow$ outstanding-refs is-volatile-Write$_{sb}$ sb ={}

by auto

from buffered-val-witness [OF buf-v] **obtain** volatile′ sop′ A′ L′ R′ W′
where
witness: $\text{Write}_{\mathsf{sb}}$ volatile′ a sop′ v A′ L′ R′ W′ ∈ set sb
by auto

{
fix j p_j is_{sbj} $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_{\mathsf{sbj}}$ j_{sbj} sb_j
assume j-bound: j < length ts_{sb}
assume neq-i-j: i ≠ j
assume jth: ts_{sb}!j = (p_j,is_{sbj}, j_{sbj}, sb_j, $\mathcal{D}_{\mathsf{sbj}}$, $\mathcal{O}_j$,$\mathcal{R}_j$)
assume non-vol: ¬ volatile
have a ∉ $\mathcal{O}_j$ ∪ all-acquired sb_j
proof
  assume a-j: a ∈ $\mathcal{O}_j$ ∪ all-acquired sb_j
  let ?take-sb_j = (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_j)
  let ?drop-sb_j = (dropWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_j)

from ts-sim [rule-format, OF j-bound] jth
  obtain suspends_j is_j $\mathcal{D}_j$ **where**
    suspends_j: suspends_j = ?drop-sb_j **and**
    is_j: instrs suspends_j @ is_{sbj} = is_j @ prog-instrs suspends_j **and**
    $\mathcal{D}_j$: $\mathcal{D}_{\mathsf{sbj}}$ = ($\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$\text{Write}_{\mathsf{sb}}$ sb_j ≠ {}) **and**
    ts_j: ts!j = (hd-prog p_j suspends_j, is_j,
    j_{sbj} |‘ (dom j_{sbj} − read-tmps suspends_j),(),
    $\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$,release ?take-sb_j (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}_j$)
    by (auto simp add: Let-def)

  from a-j ownership-distinct [OF i-bound j-bound neq-i-j ts_{sb}-i jth]
  have a-notin-sb: a ∉ $\mathcal{O}_{\mathsf{sb}}$ ∪ all-acquired sb
    by auto
  with acquired-all-acquired [of True sb $\mathcal{O}_{\mathsf{sb}}$]
  have a-not-acq: a ∉ acquired True sb $\mathcal{O}_{\mathsf{sb}}$ **by** blast
  with access-cond′ non-vol
  have a-ro: a ∈ read-only (share ?drop-sb $\mathcal{S}$)
    by auto
      from read-only-share-unowned-in [OF weak-consis-drop a-ro] a-notin-sb
        acquired-all-acquired [of True ?take-sb $\mathcal{O}_{\mathsf{sb}}$]
        all-acquired-append [of ?take-sb ?drop-sb]
      have a-ro-shared: a ∈ read-only $\mathcal{S}$
        by auto

      from rels-nv-cond [rule-format, OF non-vol j-bound [simplified leq] neq-i-j] ts_j
      have a ∉ dom (release ?take-sb_j (dom ($\mathcal{S}_{\mathsf{sb}}$)) $\mathcal{R}_j$)
        by auto

with dom-release-takeWhile [of $\mathrm{sb_j}$ (dom ($\mathcal{S}_{\mathsf{sb}}$)) $\mathcal{R}_{\mathsf{j}}$]
obtain
a-rels$_{\mathsf{j}}$: a $\notin$ dom $\mathcal{R}_{\mathsf{j}}$ **and**
a-shared$_{\mathsf{j}}$: a $\notin$ all-shared ?take-sb$_{\mathsf{j}}$
by auto

have a $\notin \bigcup((\lambda(\text{-}, \text{-}, \text{-}, \mathrm{sb}, \text{-}, \text{-}, \text{-})$. all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb)) ‘
set ts$_{\mathsf{sb}}$)
proof –
{
fix k $\mathrm{p_k}$ $\mathrm{is_k}$ $\mathrm{j_k}$ $\mathrm{sb_k}$ $\mathcal{D}_{\mathsf{k}}$ $\mathcal{O}_{\mathsf{k}}$ $\mathcal{R}_{\mathsf{k}}$
assume k-bound: k < length ts$_{\mathsf{sb}}$
assume ts-k: ts$_{\mathsf{sb}}$! k = ($\mathrm{p_k}$,$\mathrm{is_k}$,$\mathrm{j_k}$,$\mathrm{sb_k}$,$\mathcal{D}_{\mathsf{k}}$,$\mathcal{O}_{\mathsf{k}}$,$\mathcal{R}_{\mathsf{k}}$)
assume a-in: a $\in$ all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) $\mathrm{sb_k}$)
have False
proof (cases k=j)
case True **with** a-shared$_{\mathsf{j}}$ jth ts-k a-in **show** False **by** auto
next
case False
from ownership-distinct [OF j-bound k-bound False [symmetric] jth ts-k] a-j
have a $\notin$ ($\mathcal{O}_{\mathsf{k}}$ $\cup$ all-acquired $\mathrm{sb_k}$) **by** auto
with all-shared-acquired-or-owned [OF sharing-consis [OF k-bound ts-k]] a-in
show False
using all-acquired-append [of takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) $\mathrm{sb_k}$
dropWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) $\mathrm{sb_k}$]
all-shared-append [of takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) $\mathrm{sb_k}$
dropWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) $\mathrm{sb_k}$] **by** auto
qed
}
thus ?thesis **by** (fastforce simp add: in-set-conv-nth)
qed
with a-ro-shared
read-only-shared-all-until-volatile-write-subset$'$ [of ts$_{\mathsf{sb}}$ $\mathcal{S}_{\mathsf{sb}}$]
have a-ro-shared$_{\mathsf{sb}}$: a $\in$ read-only $\mathcal{S}_{\mathsf{sb}}$
by (auto simp add: $\mathcal{S}$)

with read-only-unowned [OF j-bound jth]
have a-notin-owns-j: a $\notin$ $\mathcal{O}_{\mathsf{j}}$
by auto

have own-dist: ownership-distinct ts$_{\mathsf{sb}}$ **by** fact
have share-consis: sharing-consis $\mathcal{S}_{\mathsf{sb}}$ ts$_{\mathsf{sb}}$ **by** fact
from sharing-consistent-share-all-until-volatile-write [OF own-dist share-consis i-bound ts$_{\mathsf{sb}}$-i]
have consis$'$: sharing-consistent $\mathcal{S}$ (acquired True ?take-sb $\mathcal{O}_{\mathsf{sb}}$) ?drop-sb
by (simp add: $\mathcal{S}$)

from share-all-until-volatile-write-thread-local [OF own-dist share-consis j-bound jth a-j] a-ro-shared
have a-ro-take: a $\in$ read-only (share ?take-sb$_j$ $\mathcal{S}_{sb}$)
by (auto simp add: domIff $\mathcal{S}$ read-only-def)
from sharing-consis [OF j-bound jth]
have sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb$_j$.
from sharing-consistent-weak-sharing-consistent [OF this] weak-sharing-consistent-append [of $\mathcal{O}_j$?take-sb$_j$?drop-sb$_j$]
have weak-consis-drop:weak-sharing-consistent $\mathcal{O}_j$?take-sb$_j$
by auto
from read-only-share-acquired-all-shared [OF this read-only-unowned [OF j-bound jth] a-ro-take] a-notin-owns-j a-shared$_j$
have a $\notin$ all-acquired ?take-sb$_j$
by auto
with a-j a-notin-owns-j
have a-drop: a $\in$ all-acquired ?drop-sb$_j$
using all-acquired-append [of ?take-sb$_j$?drop-sb$_j$]
by simp

from i-bound j-bound leq **have** j-bound-ts$'$: j < length ?ts$'$
by auto

note conflict-drop = a-drop [simplified suspends$_j$ [symmetric]]
from split-all-acquired-in [OF conflict-drop]

show False
proof
assume $\exists$sop a$'$ v ys zs A L R W.
(suspends$_j$ = ys @ Write$_{sb}$ True a$'$ sop v A L R W# zs) $\wedge$ a $\in$ A
then
obtain a$'$ sop$'$ v$'$ ys zs A$'$ L$'$ R$'$ W$'$ **where**
split-suspends$_j$: suspends$_j$ = ys @ Write$_{sb}$ True a$'$ sop$'$ v$'$ A$'$ L$'$ R$'$ W$'$# zs
(**is** suspends$_j$ = ?suspends) **and**
a-A$'$: a $\in$ A$'$
by blast

from sharing-consis [OF j-bound jth]
have sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb$_j$.
then have A$'$-R$'$: A$'$ $\cap$ R$'$ = {}
by (simp add: sharing-consistent-append [of - - ?take-sb$_j$?drop-sb$_j$, simplified] suspends$_j$ [symmetric] split-suspends$_j$ sharing-consistent-append)
from valid-program-history [OF j-bound jth]
have causal-program-history is$_{sbj}$ sb$_j$.
then have cph: causal-program-history is$_{sbj}$?suspends
apply –
apply (rule causal-program-history-suffix [**where** sb=?take-sb$_j$])
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
apply (simp add: split-suspends$_j$)
done

from ts_j neq-i-j j-bound
have ts'-j: ?$ts'!j$ = (hd-prog p_j $suspends_j$, is_j,
j_{sbj} |' (dom j_{sbj} − read-tmps $suspends_j$),(),
$\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$, release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
by auto
from valid-last-prog [OF j-bound jth] **have** last-prog: last-prog p_j sb_j = p_j.
then
have lp: last-prog p_j $suspends_j$ = p_j
apply −
apply (rule last-prog-same-append [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done
from valid-reads [OF j-bound jth]
have reads-consis-j: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.
from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts_{sb}› j-bound
jth reads-consis-j]
have reads-consis-m-j: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
by (simp add: m $suspends_j$)

from outstanding-non-write-non-vol-reads-drop-disj [OF i-bound j-bound neq-i-j ts_{sb}-i jth]
have outstanding-refs is-$Write_{sb}$?drop-sb ∩ outstanding-refs is-non-volatile-$Read_{sb}$ $suspends_j$ = {}
by (simp add: $suspends_j$)
from reads-consistent-flush-independent [OF this reads-consis-m-j]
have reads-consis-flush-suspend: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$)
(flush ?drop-sb m) $suspends_j$.
hence reads-consis-ys: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$)
(flush ?drop-sb m) (ys@[$Write_{sb}$ True a' sop' v' A' L' R' W'])
by (simp add: split-$suspends_j$ reads-consistent-append)

from valid-write-sops [OF j-bound jth]
have ∀sop∈write-sops (?take-sb_j@?suspends). valid-sop sop
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain valid-sops-take: ∀sop∈write-sops ?take-sb_j. valid-sop sop **and**
valid-sops-drop: ∀sop∈write-sops (ys@[$Write_{sb}$ True a' sop' v' A' L' R' W']). valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound jth]
have distinct-read-tmps (?take-sb_j@$suspends_j$)
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb_j ∩ read-tmps $suspends_j$ = {} **and**

distinct-read-tmps-drop: distinct-read-tmps suspends_j
apply (simp only: split-suspends_j [symmetric] suspends_j)
apply (simp only: distinct-read-tmps-append)
done

from valid-history [OF j-bound jth]
have h-consis:
history-consistent j_{sbj} (hd-prog p_j (?take-sb_j@suspends_j)) (?take-sb_j@suspends_j)
apply (simp only: split-suspends_j [symmetric] suspends_j)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j suspends_j)
proof −
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j suspends_j) ?take-sb_j = hd-prog p_j suspends_j
by (simp only: split-suspends_j [symmetric] suspends_j)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ suspends_j)) ?take-sb_j =
last-prog (hd-prog p_j suspends_j) ?take-sb_j
apply (simp only: split-suspends_j [symmetric] suspends_j)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-suspends_j [symmetric] suspends_j)
qed

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop
h-consis] last-prog-hd-prog
have hist-consis′: history-consistent j_{sbj} (hd-prog p_j suspends_j) suspends_j
by (simp add: split-suspends_j [symmetric] suspends_j)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis-j]
have no-vol-read: outstanding-refs is-volatile-Read_{sb}
(ys@[Write_{sb} True a′ sop′ v′ A′ L′ R′ W′]) = {}
by (auto simp add: outstanding-refs-append suspends_j [symmetric]
split-suspends_j)

have acq-simp:
acquired True (ys @ [Write_{sb} True a′ sop′ v′ A′ L′ R′ W′])
(acquired True ?take-sb_j $\mathcal{O}_j$) =
acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$) ∪ A′ − R′
by (simp add: acquired-append)

from flush-store-buffer-append [**where** sb=ys@[Write_{sb} True a′ sop′ v′ A′ L′ R′ W′]
and sb′=zs, simplified,
OF j-bound-ts′ is_j [simplified split-suspends_j] cph [simplified suspends_j]
ts′-j [simplified split-suspends_j] refl lp [simplified split-suspends_j] reads-consis-ys
hist-consis′ [simplified split-suspends_j] valid-sops-drop

distinct-read-tmps-drop [simplified split-suspends$_j$]
no-volatile-Read$_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=share ?drop-sb $\mathcal{S}$]

obtain is$_j$' $\mathcal{R}_j$' **where**
is$_j$': instrs zs @ is$_{sbj}$ = is$_j$' @ prog-instrs zs **and**
steps-ys: (?ts', flush ?drop-sb m, share ?drop-sb $\mathcal{S}$) $\Rightarrow_d$*
(?ts'[j:=(last-prog
(hd-prog p$_j$ (Write$_{sb}$ True a' sop' v' A' L' R' W'# zs)) (ys@[Write$_{sb}$ True a' sop' v' A' L' R' W']),
is$_j$',
j$_{sbj}$ |' (dom j$_{sbj}$ − read-tmps zs),
(), True, acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ∪ A' − R',$\mathcal{R}_j$')],
flush (ys@[Write$_{sb}$ True a' sop' v' A' L' R' W']) (flush ?drop-sb m),
share (ys@[Write$_{sb}$ True a' sop' v' A' L' R' W']) (share ?drop-sb $\mathcal{S}$))
(**is** (-,-,-) $\Rightarrow_d$* (?ts-ys,?m-ys,?shared-ys))

by (auto simp add: acquired-append outstanding-refs-append)

from i-bound' **have** i-bound-ys: i < length ?ts-ys
by auto

from i-bound' neq-i-j
have ts-ys-i: ?ts-ys!i = (p$_{sb}$, is$_{sb}$, j$_{sb}$,(),
$\mathcal{D}_{sb}$, acquired True sb $\mathcal{O}_{sb}$, release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
by simp
note conflict-computation = rtranclp-trans [OF steps-flush-sb steps-ys]

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys).

from safe-delayedE [OF this i-bound-ys ts-ys-i, simplified is$_{sb}$] non-vol a-not-acq
have a ∈ read-only (share (ys@[Write$_{sb}$ True a' sop' v' A' L' R' W']) (share ?drop-sb $\mathcal{S}$))
apply cases
apply (auto simp add: Let-def is$_{sb}$)
done

with a-A'
show False
by (simp add: share-append in-read-only-convs)
next
assume ∃ A L R W ys zs. suspends$_j$ = ys @ Ghost$_{sb}$ A L R W # zs ∧ a ∈ A
then
obtain A' L' R' W' ys zs **where**
split-suspends$_j$: suspends$_j$ = ys @ Ghost$_{sb}$ A' L' R' W'# zs
(**is** suspends$_j$ = ?suspends) **and**
a-A': a ∈ A'
by blast

from valid-program-history [OF j-bound jth]
have causal-program-history is_{sbj} sb_j.
then have cph: causal-program-history is_{sbj} ?suspends
  apply –
  apply (rule causal-program-history-suffix [**where** sb=?take-sb_j])
  apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
  apply (simp add: split-$suspends_j$)
  done

from ts_j neq-i-j j-bound
have ts′-j: ?ts′!j = (hd-prog p_j $suspends_j$, is_j,
  j_{sbj} |' (dom j_{sbj} – read-tmps $suspends_j$),(),
  $\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$, release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
  by auto
from valid-last-prog [OF j-bound jth] **have** last-prog: last-prog p_j sb_j = p_j.
then
have lp: last-prog p_j $suspends_j$ = p_j
  apply –
  apply (rule last-prog-same-append [**where** sb=?take-sb_j])
  apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
  apply simp
  done

from valid-reads [OF j-bound jth]
have reads-consis-j: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.
from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts_{sb}› j-bound
  jth reads-consis-j]
have reads-consis-m-j: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
  by (simp add: m $suspends_j$)

from outstanding-non-write-non-vol-reads-drop-disj [OF i-bound j-bound neq-i-j ts_{sb}-i jth]
have outstanding-refs is-$Write_{sb}$?drop-sb ∩ outstanding-refs is-non-volatile-$Read_{sb}$ $suspends_j$ = {}
  by (simp add: $suspends_j$)
from reads-consistent-flush-independent [OF this reads-consis-m-j]
have reads-consis-flush-suspend: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$)
  (flush ?drop-sb m) $suspends_j$.

hence reads-consis-ys: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$)
  (flush ?drop-sb m) (ys@[$Ghost_{sb}$ A′ L′ R′ W′])
  by (simp add: split-$suspends_j$ reads-consistent-append)
from valid-write-sops [OF j-bound jth]
have ∀sop∈write-sops (?take-sb_j@?suspends). valid-sop sop
  by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain valid-sops-take: ∀sop∈write-sops ?take-sb_j. valid-sop sop **and**
  valid-sops-drop: ∀sop∈write-sops (ys@[$Ghost_{sb}$ A′ L′ R′ W′]). valid-sop sop
  apply (simp only: write-sops-append)

apply auto
done

from read-tmps-distinct [OF j-bound jth]
have distinct-read-tmps (?take-sb_j@$suspends_j$)
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb_j $\cap$ read-tmps $suspends_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps $suspends_j$
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp only: distinct-read-tmps-append)
done

from valid-history [OF j-bound jth]
have h-consis:
history-consistent j_{sbj} (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof –
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop
h-consis] last-prog-hd-prog
have hist-consis′: history-consistent j_{sbj} (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis-j]
have no-vol-read: outstanding-refs is-volatile-$Read_{sb}$
(ys@[$Ghost_{sb}$ A′ L′ R′ W′]) = {}
by (auto simp add: outstanding-refs-append $suspends_j$ [symmetric]
split-$suspends_j$)

have acq-simp:
acquired True (ys @ [$Ghost_{sb}$ A′ L′ R′ W′])
(acquired True ?take-sb_j $\mathcal{O}_j$) =

acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$) ∪ A′ − R′
by (simp add: acquired-append)

from flush-store-buffer-append [**where** sb=ys@[$Ghost_{sb}$ A′ L′ R′ W′] **and** sb′=zs, simplified,
OF j-bound-ts′ is_j [simplified split-$suspends_j$] cph [simplified $suspends_j$]
ts′-j [simplified split-$suspends_j$] refl lp [simplified split-$suspends_j$] reads-consis-ys
hist-consis′ [simplified split-$suspends_j$] valid-sops-drop
distinct-read-tmps-drop [simplified split-$suspends_j$]
no-volatile-$Read_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=share ?drop-sb $\mathcal{S}$]
obtain is_j′ $\mathcal{R}_j$′ **where**
is_j′: instrs zs @ is_{sbj} = is_j′ @ prog-instrs zs **and**
steps-ys: (?ts′, flush ?drop-sb m, share ?drop-sb $\mathcal{S}$) $\Rightarrow_d{}^*$
(?ts′[j:=(last-prog
(hd-prog p_j ($Ghost_{sb}$ A′ L′ R′ W′# zs)) (ys@[$Ghost_{sb}$ A′ L′ R′ W′]),
is_j′,
j_{sbj} |‘ (dom j_{sbj} − read-tmps zs),
(),
$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ (ys @ [$Ghost_{sb}$ A′ L′ R′ W′]) ≠ {},
acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$) ∪ A′ − R′,$\mathcal{R}_j$′)],
flush (ys@[$Ghost_{sb}$ A′ L′ R′ W′]) (flush ?drop-sb m),
share (ys@[$Ghost_{sb}$ A′ L′ R′ W′]) (share ?drop-sb $\mathcal{S}$))
(**is** (-,-,-) $\Rightarrow_d{}^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto simp add: acquired-append)

from i-bound′ **have** i-bound-ys: i < length ?ts-ys
by auto

from i-bound′ neq-i-j
have ts-ys-i: ?ts-ys!i = (p_{sb}, is_{sb},j_{sb},(),
$\mathcal{D}_{sb}$, acquired True sb $\mathcal{O}_{sb}$, release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
by simp
note conflict-computation = rtranclp-trans [OF steps-flush-sb steps-ys]

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys)**.**

from safe-delayedE [OF this i-bound-ys ts-ys-i, simplified is_{sb}] non-vol a-not-acq
have a ∈ read-only (share (ys@[$Ghost_{sb}$ A′ L′ R′ W′]) (share ?drop-sb $\mathcal{S}$))
apply cases
apply (auto simp add: Let-def is_{sb})
done

with a-A′
show False
by (simp add: share-append in-read-only-convs)
qed
qed
}

note non-volatile-unowned-others = this

{
assume a-in: a ∈ read-only (share (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{S}$)
assume nv: ¬ volatile
have a ∈ read-only (share sb $\mathcal{S}_{sb}$)
proof (cases a ∈ $\mathcal{O}_{sb}$ ∪ all-acquired sb)
case True
from share-all-until-volatile-write-thread-local′ [OF ownership-distinct-ts$_{sb}$
sharing-consis-ts$_{sb}$ i-bound ts$_{sb}$-i True] True a-in
show ?thesis
by (simp add: $\mathcal{S}$ read-only-def)
next
case False
from read-only-share-unowned [OF weak-consis-drop - a-in] False
acquired-all-acquired [of True ?take-sb $\mathcal{O}_{sb}$] all-acquired-append [of ?take-sb
?drop-sb]
have a-ro-shared: a ∈ read-only $\mathcal{S}$
by auto

have a ∉ $\bigcup$((λ(-, -, -, sb, -, -, -).
all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)) ‘ set ts$_{sb}$)
proof −
{
fix k p$_k$ is$_k$ j$_k$ sb$_k$ $\mathcal{D}_k$ $\mathcal{O}_k$ $\mathcal{R}_k$
assume k-bound: k < length ts$_{sb}$
assume ts-k: ts$_{sb}$! k = (p$_k$,is$_k$,j$_k$,sb$_k$,$\mathcal{D}_k$,$\mathcal{O}_k$,$\mathcal{R}_k$)
assume a-in: a ∈ all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_k$)
have False
proof (cases k=i)
case True **with** False ts$_{sb}$-i ts-k a-in
all-shared-acquired-or-owned [OF sharing-consis [OF k-bound ts-k]]
all-shared-append [of takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_k$
dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_k$] **show** False **by** auto
next
case False
from rels-nv-cond [rule-format, OF nv k-bound [simplified leq] False [symmetric]
]
ts-sim [rule-format, OF k-bound] ts-k
have a ∉ dom (release (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_k$) (dom ($\mathcal{S}_{sb}$))
$\mathcal{R}_k$)
by (auto simp add: Let-def)
with dom-release-takeWhile [of sb$_k$ (dom ($\mathcal{S}_{sb}$)) $\mathcal{R}_k$]
obtain
a-rels$_j$: a ∉ dom $\mathcal{R}_k$ **and**
a-shared$_j$: a ∉ all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_k$)
by auto
with False a-in **show** ?thesis
by auto
qed

}
thus ?thesis **by** (fastforce simp add: in-set-conv-nth)
qed
with read-only-shared-all-until-volatile-write-subset$'$ [of ts_{sb} $\mathcal{S}_{sb}$] a-ro-shared
have a $\in$ read-only $\mathcal{S}_{sb}$
by (auto simp add: $\mathcal{S}$)
from read-only-share-unowned$'$ [OF weak-consis-sb read-only-unowned [OF i-bound ts_{sb}-i] False this]
show ?thesis **.**
qed
} **note** non-vol-ro-reduction = this

have valid-own$'$: valid-ownership $\mathcal{S}_{sb}'$ ts_{sb}'
proof (intro-locales)
show outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}_{sb}'$ ts_{sb}'
proof (cases volatile)
case False
from outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ts_{sb}-i]
have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ sb**.**
then

have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ (sb@[$Read_{sb}$ False a t v])
using access-cond$'$ False non-vol-ro-reduction
by (auto simp add: non-volatile-owned-or-read-only-append)

from outstanding-non-volatile-refs-owned-or-read-only-nth-update [OF i-bound this]
show ?thesis **by** (auto simp add: False ts_{sb}' sb$'$ $\mathcal{O}_{sb}'$ $\mathcal{S}_{sb}'$)
next
case True
from outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ts_{sb}-i]
have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ sb**.**
then
have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ (sb@[$Read_{sb}$ True a t v])
using True
by (simp add: non-volatile-owned-or-read-only-append)
from outstanding-non-volatile-refs-owned-or-read-only-nth-update [OF i-bound this]
show ?thesis **by** (auto simp add: True ts_{sb}' sb$'$ $\mathcal{O}_{sb}'$ $\mathcal{S}_{sb}'$)
qed
next
show outstanding-volatile-writes-unowned-by-others ts_{sb}'
proof –
have out: outstanding-refs is-volatile-$Write_{sb}$ (sb @ [$Read_{sb}$ volatile a t v]) $\subseteq$
outstanding-refs is-volatile-$Write_{sb}$ sb
by (auto simp add: outstanding-refs-append)
have all-acquired (sb @ [$Read_{sb}$ volatile a t v]) $\subseteq$ all-acquired sb
by (auto simp add: all-acquired-append)
from outstanding-volatile-writes-unowned-by-others-store-buffer
[OF i-bound ts_{sb}-i out this]
show ?thesis **by** (simp add: ts_{sb}' sb$'$ $\mathcal{O}_{sb}'$)
qed

next
show read-only-reads-unowned ts_{sb}'
proof (cases volatile)
case True
have r: read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) (sb @ [Read$_{sb}$ volatile a t v])) $\mathcal{O}_{sb}$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) (sb @ [Read$_{sb}$ volatile a t v]))
$\subseteq$ read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{O}_{sb}$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
apply (case-tac outstanding-refs (is-volatile-Write$_{sb}$) sb = {})
apply (simp-all add: outstanding-vol-write-take-drop-appends
acquired-append read-only-reads-append True)
done

have $\mathcal{O}_{sb}$ ∪ all-acquired (sb @ [Read$_{sb}$ volatile a t v]) ⊆ $\mathcal{O}_{sb}$ ∪ all-acquired sb
by (simp add: all-acquired-append)

from read-only-reads-unowned-nth-update [OF i-bound ts_{sb}-i r this]
show ?thesis
by (simp add: ts_{sb}' $\mathcal{O}_{sb}'$ sb′)
next
case False
show ?thesis
proof (unfold-locales)
fix n m
fix p_n is_n $\mathcal{O}_n$ $\mathcal{R}_n$ $\mathcal{D}_n$ j_n sb_n p_m is_m $\mathcal{O}_m$ $\mathcal{R}_m$ $\mathcal{D}_m$ j_m sb_m
assume n-bound: n < length ts_{sb}'
and m-bound: m < length ts_{sb}'
and neq-n-m: n≠m
and nth: ts_{sb}'!n = (p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$, $\mathcal{R}_n$)
and mth: ts_{sb}'!m =(p_m, is_m, j_m, sb_m, $\mathcal{D}_m$, $\mathcal{O}_m$, $\mathcal{R}_m$)
from n-bound **have** n-bound′: n < length ts_{sb} **by** (simp add: ts_{sb}')
from m-bound **have** m-bound′: m < length ts_{sb} **by** (simp add: ts_{sb}')

have acq-eq: ($\mathcal{O}_{sb}'$ ∪ all-acquired sb′) = ($\mathcal{O}_{sb}$ ∪ all-acquired sb)
by (simp add: all-acquired-append sb′ $\mathcal{O}_{sb}'$)

show ($\mathcal{O}_m$ ∪ all-acquired sb_m) ∩
read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_n) $\mathcal{O}_n$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb_n) =
{}
proof (cases m=i)
case True
with neq-n-m **have** neq-n-i: n≠i
by auto

with n-bound nth i-bound **have** nth′: ts_{sb}!n =(p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$, $\mathcal{R}_n$)
by (auto simp add: ts_{sb}')

note read-only-reads-unowned [OF n-bound′ i-bound neq-n-i nth′ ts_{sb}-i]
moreover
note acq-eq
ultimately show ?thesis
using True ts_{sb}-i nth mth n-bound′ m-bound′
by (simp add: ts_{sb}′)
next
case False
note neq-m-i = this
with m-bound mth i-bound **have** mth′: ts_{sb}!m = (p_m, is_m, j_m, sb_m, $\mathcal{D}_m$, $\mathcal{O}_m$,$\mathcal{R}_m$)
by (auto simp add: ts_{sb}′)
show ?thesis
proof (cases n=i)
case True
note read-only-reads-unowned [OF i-bound m-bound′ neq-m-i [symmetric] ts_{sb}-i mth′]
moreover
note acq-eq
moreover
note non-volatile-unowned-others [OF m-bound′ neq-m-i [symmetric] mth′]
ultimately show ?thesis
using True ts_{sb}-i nth mth n-bound′ m-bound′ neq-m-i
apply (case-tac outstanding-refs (is-volatile-Write$_{sb}$) sb = {})
apply (clarsimp simp add: outstanding-vol-write-take-drop-appends
acquired-append read-only-reads-append ts_{sb}′ sb′ $\mathcal{O}_{sb}$′)+
done
next
case False
with n-bound nth i-bound **have** nth′: ts_{sb}!n =(p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$, $\mathcal{R}_n$)
by (auto simp add: ts_{sb}′)
from read-only-reads-unowned [OF n-bound′ m-bound′ neq-n-m nth′ mth′] False neq-m-i
show ?thesis
by (clarsimp)
qed
qed
qed
qed
next
show ownership-distinct ts_{sb}′
proof −
have all-acquired (sb @ [Read$_{sb}$ volatile a t v]) ⊆ all-acquired sb
by (auto simp add: all-acquired-append)
from ownership-distinct-instructions-read-value-store-buffer-independent
[OF i-bound ts_{sb}-i this]
show ?thesis **by** (simp add: ts_{sb}′ sb′ $\mathcal{O}_{sb}$′)
qed
qed

have valid-hist′: valid-history program-step ts_{sb}′
proof −

from valid-history [OF i-bound ts_{sb}-i]
have hcons: history-consistent j_{sb} (hd-prog p_{sb} sb) sb**.**
from load-tmps-read-tmps-distinct [OF i-bound ts_{sb}-i]
have t-notin-reads: t $\notin$ read-tmps sb
  by (auto simp add: is_{sb})
from load-tmps-write-tmps-distinct [OF i-bound ts_{sb}-i]
have t-notin-writes: t $\notin \bigcup$(fst ‘ write-sops sb)
  by (auto simp add: is_{sb})

from valid-write-sops [OF i-bound ts_{sb}-i]
have valid-sops: $\forall$sop $\in$ write-sops sb. valid-sop sop
  by auto
from load-tmps-fresh [OF i-bound ts_{sb}-i]
have t-fresh: t $\notin$ dom j_{sb}
  using is_{sb}
  by simp
have history-consistent (j_{sb}(t$\mapsto$v))
    (hd-prog p_{sb} (sb@ [$Read_{sb}$ volatile a t v])) (sb@ [$Read_{sb}$ volatile a t v])
  using t-notin-writes valid-sops t-fresh hcons
  valid-implies-valid-prog-hd [OF i-bound ts_{sb}-i valid]
  apply −
  apply (rule history-consistent-appendI)
  apply (auto simp add: hd-prog-append-$Read_{sb}$)
  done
from valid-history-nth-update [OF i-bound this]
show ?thesis
  by (auto simp add: $ts_{sb}{}'$ sb$'$ $\mathcal{O}_{sb}{}'$ $j_{sb}{}'$)
    qed

    from reads-consistent-buffered-snoc [OF buf-v valid-reads [OF i-bound ts_{sb}-i] volatile-cond]
    have reads-consis$'$: reads-consistent False $\mathcal{O}_{sb}$ m_{sb} (sb @ [$Read_{sb}$ volatile a t v]) **by** (simp split: if-split-asm)

    from valid-reads-nth-update [OF i-bound this]
    have valid-reads$'$: valid-reads m_{sb} $ts_{sb}{}'$ **by** (simp add: $ts_{sb}{}'$ sb$'$ $\mathcal{O}_{sb}{}'$)

    have valid-sharing$'$: valid-sharing $\mathcal{S}_{sb}{}'$ $ts_{sb}{}'$
    proof (intro-locales)
from outstanding-non-volatile-writes-unshared [OF i-bound ts_{sb}-i]
have non-volatile-writes-unshared $\mathcal{S}_{sb}$ (sb @ [$Read_{sb}$ volatile a t v])
  by (auto simp add: non-volatile-writes-unshared-append)
from outstanding-non-volatile-writes-unshared-nth-update [OF i-bound this]
show outstanding-non-volatile-writes-unshared $\mathcal{S}_{sb}{}'$ $ts_{sb}{}'$
  by (simp add: $ts_{sb}{}'$ sb$'$ $\mathcal{S}_{sb}{}'$)
    next
from sharing-consis [OF i-bound ts_{sb}-i]
have sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ sb**.**
then
have sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ (sb @ [$Read_{sb}$ volatile a t v])

by (simp add: sharing-consistent-append)
from sharing-consis-nth-update [OF i-bound this]
show sharing-consis $\mathcal{S}_{\mathsf{sb}}'$ ts_{sb}'
by (simp add: ts_{sb}' $\mathcal{O}_{\mathsf{sb}}'$ sb$'$ $\mathcal{S}_{\mathsf{sb}}'$)
next
note read-only-unowned [OF i-bound ts_{sb}-i]
from read-only-unowned-nth-update [OF i-bound this]
show read-only-unowned $\mathcal{S}_{\mathsf{sb}}'$ ts_{sb}'
by (simp add: $\mathcal{S}_{\mathsf{sb}}'$ ts_{sb}' sb$'$ $\mathcal{O}_{\mathsf{sb}}'$)
next
from unowned-shared-nth-update [OF i-bound ts_{sb}-i subset-refl]
show unowned-shared $\mathcal{S}_{\mathsf{sb}}'$ ts_{sb}' **by** (simp add: ts_{sb}' $\mathcal{O}_{\mathsf{sb}}'$ $\mathcal{S}_{\mathsf{sb}}'$)
next
from no-outstanding-write-to-read-only-memory [OF i-bound ts_{sb}-i]
have no-write-to-read-only-memory $\mathcal{S}_{\mathsf{sb}}$ sb**.**
hence no-write-to-read-only-memory $\mathcal{S}_{\mathsf{sb}}$ (sb@[$\mathrm{Read}_{\mathsf{sb}}$ volatile a t v])
by (simp add: no-write-to-read-only-memory-append)
from no-outstanding-write-to-read-only-memory-nth-update [OF i-bound this]
show no-outstanding-write-to-read-only-memory $\mathcal{S}_{\mathsf{sb}}'$ ts_{sb}'
by (simp add: ts_{sb}' $\mathcal{S}_{\mathsf{sb}}'$ sb$'$)
qed

have tmps-distinct$'$: tmps-distinct ts_{sb}'
proof (intro-locales)
from load-tmps-distinct [OF i-bound ts_{sb}-i]
have distinct-load-tmps is_{sb}'
by (auto split: instr.splits simp add: is_{sb})
from load-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}')
next
from read-tmps-distinct [OF i-bound ts_{sb}-i]
have distinct-read-tmps sb**.**
moreover
from load-tmps-read-tmps-distinct [OF i-bound ts_{sb}-i]
have t $\notin$ read-tmps sb
by (auto simp add: is_{sb})
ultimately have distinct-read-tmps (sb @ [$\mathrm{Read}_{\mathsf{sb}}$ volatile a t v])
by (auto simp add: distinct-read-tmps-append)
from read-tmps-distinct-nth-update [OF i-bound this]
show read-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}' sb$'$)
next
from load-tmps-read-tmps-distinct [OF i-bound ts_{sb}-i]
load-tmps-distinct [OF i-bound ts_{sb}-i]
have load-tmps is_{sb}' $\cap$ read-tmps (sb @ [$\mathrm{Read}_{\mathsf{sb}}$ volatile a t v]) = {}
by (clarsimp simp add: read-tmps-append is_{sb})
from load-tmps-read-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-read-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}' sb$'$)
qed

have valid-sops$'$: valid-sops ts_{sb}'

proof −
from valid-store-sops [OF i-bound ts_{sb}-i]
have valid-store-sops′: ∀ sop∈store-sops is_{sb}′. valid-sop sop
by (auto simp add: is_{sb})
from valid-write-sops [OF i-bound ts_{sb}-i]
have valid-write-sops′: ∀ sop∈write-sops (sb@ [$Read_{sb}$ volatile a t v]). valid-sop sop
by (auto simp add: write-sops-append)
from valid-sops-nth-update [OF i-bound valid-write-sops′ valid-store-sops′]
show ?thesis **by** (simp add: ts_{sb}′ sb′)
qed

have valid-dd′: valid-data-dependency ts_{sb}′
proof −
from data-dependency-consistent-instrs [OF i-bound ts_{sb}-i]
have dd-is: data-dependency-consistent-instrs (dom j_{sb}′) is_{sb}′
by (auto simp add: is_{sb} j_{sb}′)
from load-tmps-write-tmps-distinct [OF i-bound ts_{sb}-i]
have load-tmps is_{sb}′ ∩ ⋃(fst ‘ write-sops (sb@ [$Read_{sb}$ volatile a t v])) = {}
by (auto simp add: write-sops-append is_{sb})
from valid-data-dependency-nth-update [OF i-bound dd-is this]
show ?thesis **by** (simp add: ts_{sb}′ sb′)
qed

have load-tmps-fresh′: load-tmps-fresh ts_{sb}′
proof −
from load-tmps-fresh [OF i-bound ts_{sb}-i]
have load-tmps (Read volatile a t # is_{sb}′) ∩ dom j_{sb} = {}
by (simp add: is_{sb})
moreover
from load-tmps-distinct [OF i-bound ts_{sb}-i] **have** t ∉ load-tmps is_{sb}′
by (auto simp add: is_{sb})
ultimately have load-tmps is_{sb}′ ∩ dom (j_{sb}(t ↦ v)) = {}
by auto
from load-tmps-fresh-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}′ sb′ j_{sb}′)
qed

have enough-flushs′: enough-flushs ts_{sb}′
proof −
from clean-no-outstanding-volatile-$Write_{sb}$ [OF i-bound ts_{sb}-i]
have ¬ $\mathcal{D}_{sb}$ ⟶ outstanding-refs is-volatile-$Write_{sb}$ (sb@[$Read_{sb}$ volatile a t v]) = {}
by (auto simp add: outstanding-refs-append)
from enough-flushs-nth-update [OF i-bound this]
show ?thesis
by (simp add: ts_{sb}′ sb′ $\mathcal{D}_{sb}$′)
qed

have valid-program-history′: valid-program-history ts_{sb}′
proof −
from valid-program-history [OF i-bound ts_{sb}-i]

have causal-program-history is_{sb} sb **.**
then have causal′: causal-program-history is_{sb}' (sb@[$Read_{sb}$ volatile a t v])
  by (auto simp: causal-program-history-Read is_{sb})
from valid-last-prog [OF i-bound ts_{sb}-i]
have last-prog p_{sb} sb = p_{sb}**.**
hence last-prog p_{sb} (sb @ [$Read_{sb}$ volatile a t v]) = p_{sb}
  by (simp add: last-prog-append-$Read_{sb}$)

from valid-program-history-nth-update [OF i-bound causal′ this]
show ?thesis
  by (simp add: ts_{sb}' sb′)
    qed
    show ?thesis
    proof (cases outstanding-refs is-volatile-$Write_{sb}$ sb = {})
case True

from True **have** flush-all: takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb = sb
  by (auto simp add: outstanding-refs-conv)

from True **have** suspend-nothing: dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb = []
  by (auto simp add: outstanding-refs-conv)

hence suspends-empty: suspends = []
  by (simp add: suspends)
from suspends-empty is-sim **have** is: is = Read volatile a t # is_{sb}'
  by (simp add: is_{sb})
with suspends-empty ts-i
have ts-i: ts!i = (p_{sb}, Read volatile a t # is_{sb}', j_{sb},(), $\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$, release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
  by simp

from direct-memop-step.Read
have (Read volatile a t # is_{sb}', j_{sb}, (), m, $\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$,
      release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$, $\mathcal{S}$) →
    (is_{sb}', j_{sb}(t ↦ m a), (), m, $\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$,release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$, $\mathcal{S}$)**.**
from direct-computation.concurrent-step.Memop [OF i-bound′ ts-i this]
have (ts, m, $\mathcal{S}$) $\Rightarrow_d$ (ts[i := (p_{sb}, is_{sb}', j_{sb}(t ↦ m a), (),
      $\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$, release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)], m, $\mathcal{S}$) **.**

moreover

from flush-all-until-volatile-write-Read-commute [OF i-bound ts_{sb}-i [simplified is_{sb}]]
have flush-commute: flush-all-until-volatile-write
    (ts_{sb}[i := (p_{sb},is_{sb}',
      j_{sb}(t↦v), sb @ [$Read_{sb}$ volatile a t v], $\mathcal{D}_{sb}$, $\mathcal{O}_{sb}$, $\mathcal{R}_{sb}$)]) m_{sb} =
    flush-all-until-volatile-write ts_{sb} m_{sb}**.**

from True witness **have** not-volatile′: volatile′ = False
  by (auto simp add: outstanding-refs-conv)

from witness not-volatile′ **have** a-out-sb: a ∈ outstanding-refs (Not ∘ is-volatile) sb
  apply (cases sop′)
  apply (fastforce simp add: outstanding-refs-conv is-volatile-def split: memref.splits)
  done

with non-volatile-owned-or-read-only-outstanding-refs
[OF outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ts_{sb}-i]]
have a-owned: a ∈ $\mathcal{O}_{sb}$ ∪ all-acquired sb ∪ read-only-reads $\mathcal{O}_{sb}$ sb
  by auto

have flush-all-until-volatile-write ts_{sb} m_{sb} a = v
proof −
    have ∀j < length ts_{sb}. i ≠ j ⟶
        (let (-,-,-,sb_j,-,-,-) = ts_{sb}!j
            in a ∉ outstanding-refs is-non-volatile-$Write_{sb}$ (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j))
  proof −
   {
    fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs_j sb_j
    assume j-bound: j < length ts_{sb}
    assume neq-i-j: i ≠ j
    assume jth: ts_{sb}!j = (p_j,is_j, xs_j, sb_j, $\mathcal{D}_j$, $\mathcal{O}_j$, $\mathcal{R}_j$)
   have a ∉ outstanding-refs is-non-volatile-$Write_{sb}$ (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)
    proof
 let ?take-sb_j = (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)
 let ?drop-sb_j = (dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)
 assume a-in: a ∈ outstanding-refs is-non-volatile-$Write_{sb}$?take-sb_j
 with outstanding-refs-takeWhile [**where** P′= Not ∘ is-volatile-$Write_{sb}$]
 have a-in′: a ∈ outstanding-refs is-non-volatile-$Write_{sb}$ sb_j
  by auto
 with non-volatile-owned-or-read-only-outstanding-non-volatile-writes
 [OF outstanding-non-volatile-refs-owned-or-read-only [OF j-bound jth]]
 have j-owns: a ∈ $\mathcal{O}_j$ ∪ all-acquired sb_j
  by auto
 with ownership-distinct [OF i-bound j-bound neq-i-j ts_{sb}-i jth]
 have a-not-owns: a ∉ $\mathcal{O}_{sb}$ ∪ all-acquired sb
  by blast

 from non-volatile-owned-or-read-only-append [of False $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb_j ?drop-sb_j]
  outstanding-non-volatile-refs-owned-or-read-only [OF j-bound jth]
 have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb_j
  by simp
 from non-volatile-owned-or-read-only-outstanding-non-volatile-writes [OF this] a-in
 have j-owns-drop: a ∈ $\mathcal{O}_j$ ∪ all-acquired ?take-sb_j
  by auto

from rels-cond [rule-format, OF j-bound [simplified leq] neq-i-j] ts-sim [rule-format, OF j-bound] jth
have no-unsharing:release ?take-$\mathsf{sb_j}$ (dom ($\mathcal{S}_{\mathsf{sb}}$)) $\mathcal{R}_{\mathsf{j}}$ a $\neq$ Some False
by (auto simp add: Let-def)

{
assume a $\in$ acquired True sb $\mathcal{O}_{\mathsf{sb}}$
with acquired-all-acquired-in [OF this] ownership-distinct [OF i-bound j-bound neq-i-j $\mathsf{ts_{sb}}$-i jth]
j-owns
have False
by auto
}
moreover
{
assume a-ro: a $\in$ read-only (share ?drop-sb $\mathcal{S}$)

from read-only-share-unowned-in [OF weak-consis-drop a-ro] a-not-owns
acquired-all-acquired [of True ?take-sb $\mathcal{O}_{\mathsf{sb}}$]
all-acquired-append [of ?take-sb ?drop-sb]
have a $\in$ read-only $\mathcal{S}$
by auto
with share-all-until-volatile-write-thread-local [OF ownership-distinct-$\mathsf{ts_{sb}}$ sharing-consis-$\mathsf{ts_{sb}}$ j-bound jth j-owns]
have a $\in$ read-only (share ?take-$\mathsf{sb_j}$ $\mathcal{S}_{\mathsf{sb}}$)
by (auto simp add: read-only-def $\mathcal{S}$)
hence a-dom: a $\in$ dom (share ?take-$\mathsf{sb_j}$ $\mathcal{S}_{\mathsf{sb}}$)
by (auto simp add: read-only-def domIff)
from outstanding-non-volatile-writes-unshared [OF j-bound jth]
non-volatile-writes-unshared-append [of $\mathcal{S}_{\mathsf{sb}}$?take-$\mathsf{sb_j}$?drop-$\mathsf{sb_j}$]
have nvw: non-volatile-writes-unshared $\mathcal{S}_{\mathsf{sb}}$?take-$\mathsf{sb_j}$ **by** auto
from release-not-unshared-no-write-take [OF this no-unsharing a-dom] a-in
have False **by** auto
}
moreover
{
assume a-share: volatile $\wedge$ a $\in$ dom (share ?drop-sb $\mathcal{S}$)
from outstanding-non-volatile-writes-unshared [OF j-bound jth]
have non-volatile-writes-unshared $\mathcal{S}_{\mathsf{sb}}$ $\mathsf{sb_j}$.
with non-volatile-writes-unshared-append [of $\mathcal{S}_{\mathsf{sb}}$?take-$\mathsf{sb_j}$
?drop-$\mathsf{sb_j}$]
have unshared-take: non-volatile-writes-unshared $\mathcal{S}_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-$\mathsf{Write_{sb}}$) $\mathsf{sb_j}$)
by clarsimp

from valid-own **have** own-dist: ownership-distinct $\mathsf{ts_{sb}}$
by (simp add: valid-ownership-def)
from valid-sharing **have** sharing-consis $\mathcal{S}_{\mathsf{sb}}$ $\mathsf{ts_{sb}}$
by (simp add: valid-sharing-def)

from sharing-consistent-share-all-until-volatile-write [OF own-dist this i-bound ts_{sb}-i]
have sc: sharing-consistent $\mathcal{S}$ (acquired True ?take-sb $\mathcal{O}_{sb}$) ?drop-sb
by (simp add: $\mathcal{S}$)
from sharing-consistent-share-all-shared
have dom (share ?drop-sb $\mathcal{S}$) $\subseteq$ dom $\mathcal{S}$ $\cup$ all-shared ?drop-sb
by auto
also from sharing-consistent-all-shared [OF sc]
have ... $\subseteq$ dom $\mathcal{S}$ $\cup$ acquired True ?take-sb $\mathcal{O}_{sb}$ **by** auto
also from acquired-all-acquired all-acquired-takeWhile
have ... $\subseteq$ dom $\mathcal{S}$ $\cup$ ($\mathcal{O}_{sb}$ $\cup$ all-acquired sb) **by** force
finally
have a-shared: a $\in$ dom $\mathcal{S}$
using a-share a-not-owns
by auto

with share-all-until-volatile-write-thread-local [OF ownership-distinct-ts_{sb} sharing-consis-ts_{sb} j-bound jth j-owns]
have a-dom: a $\in$ dom (share ?take-sb_j $\mathcal{S}_{sb}$)
by (auto simp add: $\mathcal{S}$ domIff)
from release-not-unshared-no-write-take [OF unshared-take no-unsharing a-dom] a-in
have False **by** auto

}
ultimately show False
using access-cond′
by auto
qed
}
thus ?thesis
by (fastforce simp add: Let-def)
qed

from flush-all-until-volatile-write-buffered-val-conv
[OF True i-bound ts_{sb}-i this]
show ?thesis
by (simp add: buf-v)
qed

hence m-a-v: m a = v
by (simp add: m)

have tmps-commute: j_{sb}(t $\mapsto$ v) = (j_{sb} |‘ (dom j_{sb} $-$ {t}))(t $\mapsto$ v)
apply (rule ext)
apply (auto simp add: restrict-map-def domIff)
done

from suspend-nothing
have suspend-nothing′: (dropWhile (Not $\circ$ is-volatile-Write_{sb}) sb′) = []

  by (simp add: sb')

 from $\mathcal{D}$
 have $\mathcal{D}'$: $\mathcal{D}_{sb}$ = ($\mathcal{D}$ ∨ outstanding-refs is-volatile-Write$_{sb}$ (sb@[Read$_{sb}$ volatile a t v]) ≠ {})
  by (auto simp: outstanding-refs-append)

 have (ts_{sb}',m_{sb},$\mathcal{S}_{sb}'$) ∼ (ts[i := (p_{sb},is_{sb}',
 j_{sb}(t↦m a),(),$\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$,
 release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)],m,$\mathcal{S}$)
  apply (rule sim-config.intros)
  apply (simp add: m flush-commute ts_{sb}' $\mathcal{O}_{sb}'$ j_{sb}' sb' $\mathcal{D}_{sb}'$ $\mathcal{R}_{sb}'$)
  using share-all-until-volatile-write-Read-commute [OF i-bound ts_{sb}-i [simplified is_{sb}]]
  apply (simp add: $\mathcal{S}$ $\mathcal{S}_{sb}'$ ts_{sb}' sb' $\mathcal{O}_{sb}'$ j_{sb}' $\mathcal{R}_{sb}'$)
  using leq
  apply (simp add: ts_{sb}')
  using i-bound i-bound′ ts-sim ts-i True $\mathcal{D}'$
  apply (clarsimp simp add: Let-def nth-list-update
   outstanding-refs-conv m-a-v ts_{sb}' $\mathcal{O}_{sb}'$ $\mathcal{S}_{sb}'$ j_{sb}' sb' $\mathcal{R}_{sb}'$ suspend-nothing′
   $\mathcal{D}_{sb}'$ flush-all acquired-append release-append
   split: if-split-asm)
  apply (rule tmps-commute)
  done

 ultimately show ?thesis
  using valid-own′ valid-hist′ valid-reads′ valid-sharing′ tmps-distinct′
   valid-sops′ valid-dd′ load-tmps-fresh′ enough-flushs′
    valid-program-history′ valid′
   m_{sb}' $\mathcal{S}_{sb}'$ $\mathcal{O}_{sb}'$
  by (auto simp del: fun-upd-apply)
   next
 case False

 then obtain r **where** r-in: r ∈ set sb **and** volatile-r: is-volatile-Write$_{sb}$ r
  by (auto simp add: outstanding-refs-conv)
 from takeWhile-dropWhile-real-prefix
 [OF r-in, of (Not ∘ is-volatile-Write$_{sb}$), simplified, OF volatile-r]
 obtain a′ v′ sb″ sop′ A′ L′ R′ W′ **where**
  sb-split: sb = takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb @ Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′# sb″
  and
  drop: dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb = Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′# sb″
  apply (auto)
  subgoal for y ys
  apply (case-tac y)
  apply auto
  done
  done

from drop suspends **have** suspends: suspends = $\text{Write}_{\mathsf{sb}}$ True a′ sop′ v′ A′ L′ R′ W′#
sb″
  by simp

have (ts, m, $\mathcal{S}$) $\Rightarrow_{\mathsf{d}}^*$ (ts, m, $\mathcal{S}$) **by** auto

moreover

from flush-all-until-volatile-write-Read-commute [OF i-bound ts_{sb}-i
  [simplified is_{sb}]]

have flush-commute: flush-all-until-volatile-write
  (ts_{sb}[i := (p_{sb},is_{sb}′, j_{sb}(t $\mapsto$ v), sb @ [$\text{Read}_{\mathsf{sb}}$ volatile a t v], $\mathcal{D}_{\mathsf{sb}}$, $\mathcal{O}_{\mathsf{sb}}$, $\mathcal{R}_{\mathsf{sb}}$)]) m_{sb}
=
  flush-all-until-volatile-write ts_{sb} m_{sb}.

have $\text{Write}_{\mathsf{sb}}$ True a′ sop′ v′ A′ L′ R′ W′∈ set sb
  by (subst sb-split) auto

from dropWhile-append1 [OF this, of (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$)]
have drop-app-comm:
  (dropWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) (sb @ [$\text{Read}_{\mathsf{sb}}$ volatile a t v])) =
    dropWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb @ [$\text{Read}_{\mathsf{sb}}$ volatile a t v]
  by simp

from load-tmps-fresh [OF i-bound ts_{sb}-i]
have t $\notin$ dom j_{sb}
  by (auto simp add: is_{sb})
then have tmps-commute:
  j_{sb} |‘ (dom j_{sb} − read-tmps sb″) =
    j_{sb} |‘ (dom j_{sb} − insert t (read-tmps sb″))
  apply −
  apply (rule ext)
  apply auto
  done

from $\mathcal{D}$
have $\mathcal{D}$′: $\mathcal{D}_{\mathsf{sb}}$ = ($\mathcal{D}$ ∨ outstanding-refs is-volatile-$\text{Write}_{\mathsf{sb}}$ (sb@[$\text{Read}_{\mathsf{sb}}$ volatile a t v]) ≠
{})
  by (auto simp: outstanding-refs-append)

have (ts_{sb}′,m_{sb},$\mathcal{S}_{\mathsf{sb}}$) ∼ (ts,m,$\mathcal{S}$)
  apply (rule sim-config.intros)
  apply (simp add: m flush-commute ts_{sb}′ $\mathcal{O}_{\mathsf{sb}}$′ $\mathcal{R}_{\mathsf{sb}}$′ j_{sb}′ sb′ $\mathcal{D}_{\mathsf{sb}}$′)
  using share-all-until-volatile-write-Read-commute [OF i-bound ts_{sb}-i [simplified is_{sb}]]
  apply (simp add: $\mathcal{S}$ $\mathcal{S}_{\mathsf{sb}}$′ ts_{sb}′ sb′ $\mathcal{O}_{\mathsf{sb}}$′ $\mathcal{R}_{\mathsf{sb}}$′ j_{sb}′)
  using leq
  apply (simp add: ts_{sb}′)
  using i-bound i-bound′ ts-sim ts-i is-sim $\mathcal{D}$′

apply (clarsimp simp add: Let-def nth-list-update is-sim drop-app-comm
read-tmps-append suspends prog-instrs-append-Read$_{sb}$ instrs-append-Read$_{sb}$
hd-prog-append-Read$_{sb}$
drop is$_{sb}$ ts$_{sb}$′ sb′ $\mathcal{O}_{sb}$′ $\mathcal{R}_{sb}$′ j$_{sb}$′ $\mathcal{D}_{sb}$′ acquired-append takeWhile-append1 [OF r-in]
volatile-r
split: if-split-asm)
apply (simp add: drop tmps-commute)+
done

ultimately show ?thesis
using valid-own′ valid-hist′ valid-reads′ valid-sharing′ tmps-distinct′ valid-dd′
valid-sops′ load-tmps-fresh′ enough-flushs′
valid-program-history′ valid′ m$_{sb}$′ $\mathcal{S}_{sb}$′
by (auto simp del: fun-upd-apply)
qed
next
case (SBHReadUnbuffered a volatile t)
then obtain
is$_{sb}$: is$_{sb}$ = Read volatile a t # is$_{sb}$′ **and**
$\mathcal{O}_{sb}$′: $\mathcal{O}_{sb}$′=$\mathcal{O}_{sb}$ **and**
$\mathcal{R}_{sb}$′: $\mathcal{R}_{sb}$′=$\mathcal{R}_{sb}$ **and**
j$_{sb}$′: j$_{sb}$′ = j$_{sb}$(t↦(m$_{sb}$ a)) **and**
sb′: sb′=sb@[Read$_{sb}$ volatile a t (m$_{sb}$ a)] **and**
m$_{sb}$′: m$_{sb}$′ = m$_{sb}$ **and**
$\mathcal{S}_{sb}$′: $\mathcal{S}_{sb}$′=$\mathcal{S}_{sb}$ **and**
$\mathcal{D}_{sb}$′: $\mathcal{D}_{sb}$′=$\mathcal{D}_{sb}$ **and**
buf-None: buffered-val sb a = None

by auto

from safe-memop-flush-sb [simplified is$_{sb}$]
obtain access-cond′: a ∈ acquired True sb $\mathcal{O}_{sb}$ ∨
a ∈ read-only (share ?drop-sb $\mathcal{S}$) ∨ (volatile ∧ a ∈ dom (share ?drop-sb $\mathcal{S}$)) **and**
volatile-clean: volatile ⟶ ¬ $\mathcal{D}_{sb}$ **and**
rels-cond: ∀ j < length ts. i≠j ⟶ released (ts!j) a ≠ Some False **and**
rels-nv-cond: ¬volatile ⟶ (∀ j < length ts. i≠j ⟶ a ∉ dom (released (ts!j)))
by cases auto

from clean-no-outstanding-volatile-Write$_{sb}$ [OF i-bound ts$_{sb}$-i] volatile-clean
have volatile-cond: volatile ⟶ outstanding-refs is-volatile-Write$_{sb}$ sb ={}
by auto

{
fix j p$_j$ is$_{sbj}$ $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_{sbj}$ j$_{sbj}$ sb$_j$
assume j-bound: j < length ts$_{sb}$
assume neq-i-j: i ≠ j
assume jth: ts$_{sb}$!j = (p$_j$,is$_{sbj}$, j$_{sbj}$, sb$_j$, $\mathcal{D}_{sbj}$, $\mathcal{O}_j$,$\mathcal{R}_j$)
assume non-vol: ¬ volatile
have a ∉ $\mathcal{O}_j$ ∪ all-acquired sb$_j$

proof
  assume a-j: $a \in \mathcal{O}_j \cup$ all-acquired sb_j
  let ?take-sb_j = (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j)
  let ?drop-sb_j = (dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j)

      from ts-sim [rule-format, OF j-bound] jth
  obtain suspends$_j$ is$_j$ $\mathcal{D}_j$ **where**
    suspends$_j$: suspends$_j$ = ?drop-sb_j **and**
    is$_j$: instrs suspends$_j$ @ is$_{sbj}$ = is$_j$ @ prog-instrs suspends$_j$ **and**
    $\mathcal{D}_j$: $\mathcal{D}_{sbj} = (\mathcal{D}_j \vee$ outstanding-refs is-volatile-Write$_{sb}$ $sb_j \neq \{\})$ **and**
    ts$_j$: ts!j = (hd-prog p_j suspends$_j$, is$_j$,
    j_{sbj} |' (dom j_{sbj} − read-tmps suspends$_j$),(),
    $\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$,release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
    by (auto simp add: Let-def)

  from a-j ownership-distinct [OF i-bound j-bound neq-i-j ts$_{sb}$-i jth]
  have a-notin-sb: $a \notin \mathcal{O}_{sb} \cup$ all-acquired sb
    by auto
  with acquired-all-acquired [of True sb $\mathcal{O}_{sb}$]
  have a-not-acq: $a \notin$ acquired True sb $\mathcal{O}_{sb}$ **by** blast
  with access-cond$'$ non-vol
  have a-ro: $a \in$ read-only (share ?drop-sb $\mathcal{S}$)
    by auto
      from read-only-share-unowned-in [OF weak-consis-drop a-ro] a-notin-sb
        acquired-all-acquired [of True ?take-sb $\mathcal{O}_{sb}$]
        all-acquired-append [of ?take-sb ?drop-sb]
      have a-ro-shared: $a \in$ read-only $\mathcal{S}$
        by auto

      from rels-nv-cond [rule-format, OF non-vol j-bound [simplified leq] neq-i-j] ts$_j$
      have $a \notin$ dom (release ?take-sb_j (dom ($\mathcal{S}_{sb}$)) $\mathcal{R}_j$)
        by auto
      with dom-release-takeWhile [of sb_j (dom ($\mathcal{S}_{sb}$)) $\mathcal{R}_j$]
      obtain
        a-rels$_j$: $a \notin$ dom $\mathcal{R}_j$ **and**
        a-shared$_j$: $a \notin$ all-shared ?take-sb_j
        by auto

      have $a \notin \bigcup((\lambda(-, -, -, sb, -, -, -).$ all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$)
sb)) '
          set ts$_{sb}$)
      proof −
        {
          fix k p_k is$_k$ j_k sb_k $\mathcal{D}_k$ $\mathcal{O}_k$ $\mathcal{R}_k$
          assume k-bound: k < length ts$_{sb}$
          assume ts-k: ts$_{sb}$! k = (p_k,is$_k$,j_k,sb_k,$\mathcal{D}_k$,$\mathcal{O}_k$,$\mathcal{R}_k$)
          assume a-in: $a \in$ all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_k)

have False
proof (cases k=j)
case True with a-shared$_j$ jth ts-k a-in show False by auto
next
case False
from ownership-distinct [OF j-bound k-bound False [symmetric] jth ts-k] a-j
have a $\notin$ ($\mathcal{O}_k$ $\cup$ all-acquired sb$_k$) by auto
with all-shared-acquired-or-owned [OF sharing-consis [OF k-bound ts-k]] a-in
show False
using all-acquired-append [of takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_k$
dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_k$]
all-shared-append [of takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_k$
dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_k$] by auto
qed
}
thus ?thesis by (fastforce simp add: in-set-conv-nth)
qed
with a-ro-shared
read-only-shared-all-until-volatile-write-subset$'$ [of ts$_{sb}$ $\mathcal{S}_{sb}$]
have a-ro-shared$_{sb}$: a $\in$ read-only $\mathcal{S}_{sb}$
by (auto simp add: $\mathcal{S}$)

with read-only-unowned [OF j-bound jth]
have a-notin-owns-j: a $\notin$ $\mathcal{O}_j$
by auto

have own-dist: ownership-distinct ts$_{sb}$ by fact
have share-consis: sharing-consis $\mathcal{S}_{sb}$ ts$_{sb}$ by fact
from sharing-consistent-share-all-until-volatile-write [OF own-dist share-consis i-bound ts$_{sb}$-i]
have consis$'$: sharing-consistent $\mathcal{S}$ (acquired True ?take-sb $\mathcal{O}_{sb}$) ?drop-sb
by (simp add: $\mathcal{S}$)
from share-all-until-volatile-write-thread-local [OF own-dist share-consis j-bound jth a-j] a-ro-shared
have a-ro-take: a $\in$ read-only (share ?take-sb$_j$ $\mathcal{S}_{sb}$)
by (auto simp add: domIff $\mathcal{S}$ read-only-def)
from sharing-consis [OF j-bound jth]
have sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb$_j$.
from sharing-consistent-weak-sharing-consistent [OF this] weak-sharing-consistent-append [of $\mathcal{O}_j$?take-sb$_j$?drop-sb$_j$]
have weak-consis-drop:weak-sharing-consistent $\mathcal{O}_j$?take-sb$_j$
by auto
from read-only-share-acquired-all-shared [OF this read-only-unowned [OF j-bound jth] a-ro-take] a-notin-owns-j a-shared$_j$
have a $\notin$ all-acquired ?take-sb$_j$
by auto
with a-j a-notin-owns-j
have a-drop: a $\in$ all-acquired ?drop-sb$_j$
using all-acquired-append [of ?take-sb$_j$?drop-sb$_j$]

by simp

from i-bound j-bound leq **have** j-bound-ts′: j < length ?ts′
  by auto

note conflict-drop = a-drop [simplified suspends$_j$ [symmetric]]
from split-all-acquired-in [OF conflict-drop]

show False
proof
  assume ∃sop a′ v ys zs A L R W.
      (suspends$_j$ = ys @ Write$_{sb}$ True a′ sop v A L R W# zs) ∧ a ∈ A
  then
  obtain a′ sop′ v′ ys zs A′ L′ R′ W′ **where**
    split-suspends$_j$: suspends$_j$ = ys @ Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′# zs
    (**is** suspends$_j$ = ?suspends) **and**
a-A′: a ∈ A′
    by blast

  from sharing-consis [OF j-bound jth]
  have sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb$_j$.
  then have A′-R′: A′ ∩ R′ = {}
    by (simp add: sharing-consistent-append [of - - ?take-sb$_j$?drop-sb$_j$, simplified]
suspends$_j$ [symmetric] split-suspends$_j$ sharing-consistent-append)
  from valid-program-history [OF j-bound jth]
  have causal-program-history is$_{sbj}$ sb$_j$.
  then have cph: causal-program-history is$_{sbj}$?suspends
    apply −
    apply (rule causal-program-history-suffix [**where** sb=?take-sb$_j$])
    apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
    apply (simp add: split-suspends$_j$)
    done

  from ts$_j$ neq-i-j j-bound
  have ts′-j: ?ts′!j = (hd-prog p$_j$ suspends$_j$, is$_j$,
    j$_{sbj}$ |‘ (dom j$_{sbj}$ − read-tmps suspends$_j$),(),
    $\mathcal{D}_j$, acquired True ?take-sb$_j$ $\mathcal{O}_j$, release ?take-sb$_j$ (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
    by auto
  from valid-last-prog [OF j-bound jth] **have** last-prog: last-prog p$_j$ sb$_j$ = p$_j$.
  then
  have lp: last-prog p$_j$ suspends$_j$ = p$_j$
    apply −
    apply (rule last-prog-same-append [**where** sb=?take-sb$_j$])
    apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
    apply simp
    done
  from valid-reads [OF j-bound jth]
  have reads-consis-j: reads-consistent False $\mathcal{O}_j$ m$_{sb}$ sb$_j$.

from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{\mathsf{sb}}$ ts_{sb}› j-bound
jth reads-consis-j]
have reads-consis-m-j: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
by (simp add: m $suspends_j$)

from outstanding-non-write-non-vol-reads-drop-disj [OF i-bound j-bound neq-i-j ts_{sb}-i jth]
have outstanding-refs is-$Write_{\mathsf{sb}}$?drop-sb $\cap$ outstanding-refs is-non-volatile-$Read_{\mathsf{sb}}$ $suspends_j$ = {}
by (simp add: $suspends_j$)
from reads-consistent-flush-independent [OF this reads-consis-m-j]
have reads-consis-flush-suspend: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$)
(flush ?drop-sb m) $suspends_j$.
hence reads-consis-ys: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$)
(flush ?drop-sb m) (ys@[$Write_{\mathsf{sb}}$ True a′ sop′ v′ A′ L′ R′ W′])
by (simp add: split-$suspends_j$ reads-consistent-append)

from valid-write-sops [OF j-bound jth]
have $\forall$ sop$\in$write-sops (?take-sb_j@?suspends). valid-sop sop
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain valid-sops-take: $\forall$ sop$\in$write-sops ?take-sb_j. valid-sop sop **and**
valid-sops-drop: $\forall$ sop$\in$write-sops (ys@[$Write_{\mathsf{sb}}$ True a′ sop′ v′ A′ L′ R′ W′]). valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound jth]
have distinct-read-tmps (?take-sb_j@$suspends_j$)
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb_j $\cap$ read-tmps $suspends_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps $suspends_j$
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp only: distinct-read-tmps-append)
done

from valid-history [OF j-bound jth]
have h-consis:
history-consistent j_{sbj} (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof –
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp

from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop
h-consis] last-prog-hd-prog
have hist-consis′: history-consistent j_{sbj} (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis-j]
have no-vol-read: outstanding-refs is-volatile-$Read_{sb}$
(ys@[$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]) = {}
by (auto simp add: outstanding-refs-append $suspends_j$ [symmetric]
split-$suspends_j$)

have acq-simp:
acquired True (ys @ [$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′])
(acquired True ?take-sb_j $\mathcal{O}_j$) =
acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$) ∪ A′ − R′
by (simp add: acquired-append)

from flush-store-buffer-append [**where** sb=ys@[$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]
and sb′=zs, simplified,
OF j-bound-ts′ is_j [simplified split-$suspends_j$] cph [simplified $suspends_j$]
ts′-j [simplified split-$suspends_j$] refl lp [simplified split-$suspends_j$] reads-consis-ys
hist-consis′ [simplified split-$suspends_j$] valid-sops-drop
distinct-read-tmps-drop [simplified split-$suspends_j$]
no-volatile-$Read_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=share ?drop-sb $\mathcal{S}$]

obtain is_j′ $\mathcal{R}_j$′ **where**
is_j′: instrs zs @ is_{sbj} = is_j′ @ prog-instrs zs **and**
steps-ys: (?ts′, flush ?drop-sb m, share ?drop-sb $\mathcal{S}$) $\Rightarrow_d{}^*$
(?ts′[j:=(last-prog
(hd-prog p_j ($Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′# zs)) (ys@[$Write_{sb}$ True a′
sop′ v′ A′ L′ R′ W′]),
is_j′,
j_{sbj} |‘ (dom j_{sbj} − read-tmps zs),
(), True, acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$) ∪ A′ − R′,$\mathcal{R}_j$′)],
flush (ys@[$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]) (flush ?drop-sb m),
share (ys@[$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]) (share ?drop-sb $\mathcal{S}$))
(**is** (-,-,-) $\Rightarrow_d{}^*$ (?ts-ys,?m-ys,?shared-ys))

by (auto simp add: acquired-append outstanding-refs-append)

from i-bound′ **have** i-bound-ys: i < length ?ts-ys
by auto

from i-bound′ neq-i-j
have ts-ys-i: ?ts-ys!i = (p_{sb}, is_{sb}, j_{sb},(),
$\mathcal{D}_{\mathsf{sb}}$, acquired True sb $\mathcal{O}_{\mathsf{sb}}$, release sb (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}_{\mathsf{sb}}$)
by simp
note conflict-computation = rtranclp-trans [OF steps-flush-sb steps-ys]

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys).

from safe-delayedE [OF this i-bound-ys ts-ys-i, simplified is_{sb}] non-vol a-not-acq
have a $\in$ read-only (share (ys@[$\mathsf{Write}_{\mathsf{sb}}$ True a′ sop′ v′ A′ L′ R′ W′]) (share ?drop-sb $\mathcal{S}$))
apply cases
apply (auto simp add: Let-def is_{sb})
done

with a-A′
show False
by (simp add: share-append in-read-only-convs)
next
assume $\exists$ A L R W ys zs. $suspends_j$ = ys @ $\mathsf{Ghost}_{\mathsf{sb}}$ A L R W # zs $\wedge$ a $\in$ A
then
obtain A′ L′ R′ W′ ys zs **where**
split-$suspends_j$: $suspends_j$ = ys @ $\mathsf{Ghost}_{\mathsf{sb}}$ A′ L′ R′ W′# zs
(**is** $suspends_j$ = ?suspends) **and**
a-A′: a $\in$ A′
by blast

from valid-program-history [OF j-bound jth]
have causal-program-history is_{sbj} sb_j.
then have cph: causal-program-history is_{sbj} ?suspends
apply −
apply (rule causal-program-history-suffix [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp add: split-$suspends_j$)
done

from ts_j neq-i-j j-bound
have ts′-j: ?ts′!j = (hd-prog p_j $suspends_j$, is_j,
j_{sbj} |‘ (dom j_{sbj} − read-tmps $suspends_j$),(),
$\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$, release ?take-sb_j (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}_j$)
by auto
from valid-last-prog [OF j-bound jth] **have** last-prog: last-prog p_j sb_j = p_j.
then
have lp: last-prog p_j $suspends_j$ = p_j

apply −
apply (rule last-prog-same-append [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

from valid-reads [OF j-bound jth]
have reads-consis-j: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.
from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts_{sb}› j-bound
jth reads-consis-j]
have reads-consis-m-j: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
by (simp add: m $suspends_j$)

from outstanding-non-write-non-vol-reads-drop-disj [OF i-bound j-bound neq-i-j ts_{sb}-i jth]
have outstanding-refs is-$Write_{sb}$?drop-sb ∩ outstanding-refs is-non-volatile-$Read_{sb}$ $suspends_j$ = {}
by (simp add: $suspends_j$)
from reads-consistent-flush-independent [OF this reads-consis-m-j]
have reads-consis-flush-suspend: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$)
(flush ?drop-sb m) $suspends_j$.

hence reads-consis-ys: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$)
(flush ?drop-sb m) (ys@[$Ghost_{sb}$ A′ L′ R′ W′])
by (simp add: split-$suspends_j$ reads-consistent-append)
from valid-write-sops [OF j-bound jth]
have ∀ sop∈write-sops (?take-sb_j@?suspends). valid-sop sop
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain valid-sops-take: ∀ sop∈write-sops ?take-sb_j. valid-sop sop **and**
valid-sops-drop: ∀ sop∈write-sops (ys@[$Ghost_{sb}$ A′ L′ R′ W′]). valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound jth]
have distinct-read-tmps (?take-sb_j@$suspends_j$)
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb_j ∩ read-tmps $suspends_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps $suspends_j$
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp only: distinct-read-tmps-append)
done

from valid-history [OF j-bound jth]
have h-consis:
history-consistent j_{sbj} (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)

apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof –
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop
h-consis] last-prog-hd-prog
have hist-consis′: history-consistent j_{sbj} (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis-j]
have no-vol-read: outstanding-refs is-volatile-$Read_{sb}$
(ys@[$Ghost_{sb}$ A′ L′ R′ W′]) = {}
by (auto simp add: outstanding-refs-append $suspends_j$ [symmetric]
split-$suspends_j$)

have acq-simp:
acquired True (ys @ [$Ghost_{sb}$ A′ L′ R′ W′])
(acquired True ?take-sb_j $\mathcal{O}_j$) =
acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$) ∪ A′ – R′
by (simp add: acquired-append)

from flush-store-buffer-append [**where** sb=ys@[$Ghost_{sb}$ A′ L′ R′ W′] **and** sb′=zs,
simplified,
OF j-bound-ts′ is_j [simplified split-$suspends_j$] cph [simplified $suspends_j$]
ts′-j [simplified split-$suspends_j$] refl lp [simplified split-$suspends_j$] reads-consis-ys
hist-consis′ [simplified split-$suspends_j$] valid-sops-drop
distinct-read-tmps-drop [simplified split-$suspends_j$]
no-volatile-$Read_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=share ?drop-sb $\mathcal{S}$]
obtain is_j′ $\mathcal{R}_j$′ **where**
is_j′: instrs zs @ is_{sbj} = is_j′ @ prog-instrs zs **and**
steps-ys: (?ts′, flush ?drop-sb m, share ?drop-sb $\mathcal{S}$) $\Rightarrow_d^*$
(?ts′[j:=(last-prog
(hd-prog p_j ($Ghost_{sb}$ A′ L′ R′ W′# zs)) (ys@[$Ghost_{sb}$ A′ L′ R′ W′]),
is_j′,

j_{sbj} |` (dom j_{sbj} − read-tmps zs),
(),
$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-Write$_{sb}$ (ys @ [Ghost$_{sb}$ A′ L′ R′ W′]) ≠ {},
acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ∪ A′ − R′,$\mathcal{R}_j$′)],
flush (ys@[Ghost$_{sb}$ A′ L′ R′ W′]) (flush ?drop-sb m),
share (ys@[Ghost$_{sb}$ A′ L′ R′ W′]) (share ?drop-sb $\mathcal{S}$))
(**is** (-,-,-) $\Rightarrow_d^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto simp add: acquired-append)

from i-bound′ **have** i-bound-ys: i < length ?ts-ys
by auto

from i-bound′ neq-i-j
have ts-ys-i: ?ts-ys!i = (p_{sb}, is_{sb},j_{sb},(),
$\mathcal{D}_{sb}$, acquired True sb $\mathcal{O}_{sb}$, release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
by simp
note conflict-computation = rtranclp-trans [OF steps-flush-sb steps-ys]

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys).

from safe-delayedE [OF this i-bound-ys ts-ys-i, simplified is_{sb}] non-vol a-not-acq
have a ∈ read-only (share (ys@[Ghost$_{sb}$ A′ L′ R′ W′]) (share ?drop-sb $\mathcal{S}$))
apply cases
apply (auto simp add: Let-def is_{sb})
done

with a-A′
show False
by (simp add: share-append in-read-only-convs)
qed
qed
}
note non-volatile-unowned-others = this

{
assume a-in: a ∈ read-only (share (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{S}$)
assume nv: ¬ volatile
have a ∈ read-only (share sb $\mathcal{S}_{sb}$)
proof (cases a ∈ $\mathcal{O}_{sb}$ ∪ all-acquired sb)
case True
from share-all-until-volatile-write-thread-local′ [OF ownership-distinct-ts$_{sb}$
sharing-consis-ts$_{sb}$ i-bound ts$_{sb}$-i True] True a-in
show ?thesis
by (simp add: $\mathcal{S}$ read-only-def)
next
case False
from read-only-share-unowned [OF weak-consis-drop - a-in] False
acquired-all-acquired [of True ?take-sb $\mathcal{O}_{sb}$] all-acquired-append [of ?take-sb
?drop-sb]

```
      have a-ro-shared: a ∈ read-only 𝒮
        by auto
      have a ∉ ⋃((λ(-, -, -, sb, -, -, -).
           all-shared (takeWhile (Not ∘ is-volatile-Write_sb) sb)) ` set ts_sb)
      proof –
        {
          fix k p_k is_k j_k sb_k 𝒟_k 𝒪_k ℛ_k
          assume k-bound: k < length ts_sb
          assume ts-k: ts_sb ! k = (p_k,is_k,j_k,sb_k,𝒟_k,𝒪_k,ℛ_k)
          assume a-in: a ∈ all-shared (takeWhile (Not ∘ is-volatile-Write_sb) sb_k)
          have False
          proof (cases k=i)
            case True with False ts_sb-i ts-k a-in
              all-shared-acquired-or-owned [OF sharing-consis [OF k-bound ts-k]]
              all-shared-append [of takeWhile (Not ∘ is-volatile-Write_sb) sb_k
              dropWhile (Not ∘ is-volatile-Write_sb) sb_k] show False by auto
          next
            case False
        from rels-nv-cond [rule-format, OF nv k-bound [simplified leq] False [symmetric]
]
            ts-sim [rule-format, OF k-bound] ts-k
           have a ∉ dom (release (takeWhile (Not ∘ is-volatile-Write_sb) sb_k) (dom (𝒮_sb))
ℛ_k)
              by (auto simp add: Let-def)
            with dom-release-takeWhile [of sb_k (dom (𝒮_sb)) ℛ_k]
            obtain
              a-rels_j: a ∉ dom ℛ_k and
              a-shared_j: a ∉ all-shared (takeWhile (Not ∘ is-volatile-Write_sb) sb_k)
              by auto
            with False a-in show ?thesis
              by auto
          qed
        }
        thus ?thesis
          by (auto simp add: in-set-conv-nth)
      qed
      with read-only-shared-all-until-volatile-write-subset' [of ts_sb 𝒮_sb] a-ro-shared
      have a ∈ read-only 𝒮_sb
        by (auto simp add: 𝒮)

    from read-only-share-unowned' [OF weak-consis-sb read-only-unowned [OF i-bound
ts_sb-i] False  this]
      show ?thesis .
    qed
  } note non-vol-ro-reduction = this

  have valid-own': valid-ownership 𝒮_sb' ts_sb'
  proof (intro-locales)
show outstanding-non-volatile-refs-owned-or-read-only 𝒮_sb' ts_sb'
proof (cases volatile)
```

case False
from outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ts_{sb}-i]
have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ sb.
then

have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ (sb@[$Read_{sb}$ False a t (m_{sb} a)])
using access-cond′ False non-vol-ro-reduction
by (auto simp add: non-volatile-owned-or-read-only-append)

from outstanding-non-volatile-refs-owned-or-read-only-nth-update [OF i-bound this]
show ?thesis **by** (auto simp add: False ts_{sb}′ sb′ $\mathcal{O}_{sb}$′ $\mathcal{S}_{sb}$′)
next
case True
from outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ts_{sb}-i]
have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ sb.
then
have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ (sb@[$Read_{sb}$ True a t (m_{sb} a)])
using True
by (simp add: non-volatile-owned-or-read-only-append)
from outstanding-non-volatile-refs-owned-or-read-only-nth-update [OF i-bound this]
show ?thesis **by** (auto simp add: True ts_{sb}′ sb′ $\mathcal{O}_{sb}$′ $\mathcal{S}_{sb}$′)
qed
next
show outstanding-volatile-writes-unowned-by-others ts_{sb}′
proof −
have out: outstanding-refs is-volatile-$Write_{sb}$ (sb @ [$Read_{sb}$ volatile a t (m_{sb} a)]) ⊆
outstanding-refs is-volatile-$Write_{sb}$ sb
by (auto simp add: outstanding-refs-append)
have all-acquired (sb @ [$Read_{sb}$ volatile a t (m_{sb} a)]) ⊆ all-acquired sb
by (auto simp add: all-acquired-append)
from outstanding-volatile-writes-unowned-by-others-store-buffer
[OF i-bound ts_{sb}-i out this]
show ?thesis **by** (simp add: ts_{sb}′ sb′ $\mathcal{O}_{sb}$′)
qed
next
show read-only-reads-unowned ts_{sb}′
proof (cases volatile)
case True
have r: read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-$Write_{sb}$)
(sb @ [$Read_{sb}$ volatile a t (m_{sb} a)])) $\mathcal{O}_{sb}$)
(dropWhile (Not ∘ is-volatile-$Write_{sb}$) (sb @ [$Read_{sb}$ volatile a t (m_{sb} a)]))
⊆ read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb)
$\mathcal{O}_{sb}$)
(dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb)
apply (case-tac outstanding-refs (is-volatile-$Write_{sb}$) sb = {})
apply (simp-all add: outstanding-vol-write-take-drop-appends
acquired-append read-only-reads-append True)
done

have $\mathcal{O}_{sb}$ ∪ all-acquired (sb @ [$Read_{sb}$ volatile a t (m_{sb} a)]) ⊆ $\mathcal{O}_{sb}$ ∪ all-acquired sb

by (simp add: all-acquired-append)

from read-only-reads-unowned-nth-update [OF i-bound ts_{sb}-i r this]
show ?thesis
by (simp add: ts_{sb}' $\mathcal{O}_{sb}'$ sb′)
next
case False
show ?thesis
proof (unfold-locales)
fix n m
fix p_n is_n $\mathcal{O}_n$ $\mathcal{R}_n$ $\mathcal{D}_n$ j_n sb_n p_m is_m $\mathcal{O}_m$ $\mathcal{R}_m$ $\mathcal{D}_m$ j_m sb_m
assume n-bound: n < length ts_{sb}'
and m-bound: m < length ts_{sb}'
and neq-n-m: n≠m
and nth: ts_{sb}'!n = (p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
and mth: ts_{sb}'!m =(p_m, is_m, j_m, sb_m, $\mathcal{D}_m$, $\mathcal{O}_m$,$\mathcal{R}_m$)
from n-bound **have** n-bound′: n < length ts_{sb} **by** (simp add: ts_{sb}')
from m-bound **have** m-bound′: m < length ts_{sb} **by** (simp add: ts_{sb}')

have acq-eq: ($\mathcal{O}_{sb}'$ ∪ all-acquired sb′) = ($\mathcal{O}_{sb}$ ∪ all-acquired sb)
by (simp add: all-acquired-append sb′ $\mathcal{O}_{sb}'$)

show ($\mathcal{O}_m$ ∪ all-acquired sb_m) ∩
read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write_{sb}) sb_n) $\mathcal{O}_n$)
(dropWhile (Not ∘ is-volatile-Write_{sb}) sb_n) =
{}
proof (cases m=i)
case True
with neq-n-m **have** neq-n-i: n≠i
by auto

with n-bound nth i-bound **have** nth′: ts_{sb}!n =(p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
by (auto simp add: ts_{sb}')
note read-only-reads-unowned [OF n-bound′ i-bound neq-n-i nth′ ts_{sb}-i]
moreover
note acq-eq
ultimately show ?thesis
using True ts_{sb}-i nth mth n-bound′ m-bound′
by (simp add: ts_{sb}')
next
case False
note neq-m-i = this
with m-bound mth i-bound **have** mth′: ts_{sb}!m = (p_m, is_m, j_m, sb_m, $\mathcal{D}_m$, $\mathcal{O}_m$,$\mathcal{R}_m$)
by (auto simp add: ts_{sb}')
show ?thesis
proof (cases n=i)
case True
note read-only-reads-unowned [OF i-bound m-bound′ neq-m-i [symmetric] ts_{sb}-i mth′]
moreover

note acq-eq
moreover
note non-volatile-unowned-others [OF m-bound′ neq-m-i [symmetric] mth′]
ultimately show ?thesis
using True ts_{sb}-i nth mth n-bound′ m-bound′ neq-m-i
apply (case-tac outstanding-refs (is-volatile-Write$_{sb}$) sb = {})
apply (clarsimp simp add: outstanding-vol-write-take-drop-appends
acquired-append read-only-reads-append ts_{sb}′ sb′ $\mathcal{O}_{sb}$′)+
done
next
case False
with n-bound nth i-bound **have** nth′: ts_{sb}!n =(p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
by (auto simp add: ts_{sb}′)
from read-only-reads-unowned [OF n-bound′ m-bound′ neq-n-m nth′ mth′] False neq-m-i
show ?thesis
by (clarsimp)
qed
qed
qed
qed
show ownership-distinct ts_{sb}′
proof −
have all-acquired (sb @ [Read$_{sb}$ volatile a t (m_{sb} a)]) ⊆ all-acquired sb
by (auto simp add: all-acquired-append)
from ownership-distinct-instructions-read-value-store-buffer-independent
[OF i-bound ts_{sb}-i this]
show ?thesis **by** (simp add: ts_{sb}′ sb′ $\mathcal{O}_{sb}$′)
qed
qed

have valid-hist′: valid-history program-step ts_{sb}′
proof −
from valid-history [OF i-bound ts_{sb}-i]
have hcons: history-consistent j_{sb} (hd-prog p_{sb} sb) sb.
from load-tmps-read-tmps-distinct [OF i-bound ts_{sb}-i]
have t-notin-reads: t ∉ read-tmps sb
by (auto simp add: is_{sb})
from load-tmps-write-tmps-distinct [OF i-bound ts_{sb}-i]
have t-notin-writes: t ∉ ⋃(fst ‘ write-sops sb)
by (auto simp add: is_{sb})

from valid-write-sops [OF i-bound ts_{sb}-i]
have valid-sops: ∀sop ∈ write-sops sb. valid-sop sop
by auto
from load-tmps-fresh [OF i-bound ts_{sb}-i]
have t-fresh: t ∉ dom j_{sb}
using is_{sb}
by simp

from valid-implies-valid-prog-hd [OF i-bound ts_{sb}-i valid]
have history-consistent (j_{sb}(t$\mapsto$$m_{sb}$ a))
(hd-prog p_{sb} (sb@ [$Read_{sb}$ volatile a t (m_{sb} a)]))
(sb@ [$Read_{sb}$ volatile a t (m_{sb} a)])
using t-notin-writes valid-sops t-fresh hcons
apply –
apply (rule history-consistent-appendI)
apply (auto simp add: hd-prog-append-$Read_{sb}$)
done

from valid-history-nth-update [OF i-bound this]
show ?thesis
by (auto simp add: ts_{sb}' sb$'$ $\mathcal{O}_{sb}'$ j_{sb}')
qed

from
reads-consistent-unbuffered-snoc [OF buf-None refl valid-reads [OF i-bound ts_{sb}-i] volatile-cond]
have reads-consis$'$: reads-consistent False $\mathcal{O}_{sb}$ m_{sb} (sb @ [$Read_{sb}$ volatile a t (m_{sb} a)])
by (simp split: if-split-asm)

from valid-reads-nth-update [OF i-bound this]
have valid-reads$'$: valid-reads m_{sb} ts_{sb}' **by** (simp add: ts_{sb}' sb$'$ $\mathcal{O}_{sb}'$)

have valid-sharing$'$: valid-sharing $\mathcal{S}_{sb}'$ ts_{sb}'
proof (intro-locales)
from outstanding-non-volatile-writes-unshared [OF i-bound ts_{sb}-i]
have non-volatile-writes-unshared $\mathcal{S}_{sb}$ (sb @ [$Read_{sb}$ volatile a t (m_{sb} a)])
by (auto simp add: non-volatile-writes-unshared-append)
from outstanding-non-volatile-writes-unshared-nth-update [OF i-bound this]
show outstanding-non-volatile-writes-unshared $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: ts_{sb}' sb$'$ $\mathcal{S}_{sb}'$)
next
from sharing-consis [OF i-bound ts_{sb}-i]
have sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ sb.
then
have sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ (sb @ [$Read_{sb}$ volatile a t (m_{sb} a)])
by (simp add: sharing-consistent-append)
from sharing-consis-nth-update [OF i-bound this]
show sharing-consis $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: ts_{sb}' $\mathcal{O}_{sb}'$ sb$'$ $\mathcal{S}_{sb}'$)
next
note read-only-unowned [OF i-bound ts_{sb}-i]
from read-only-unowned-nth-update [OF i-bound this]
show read-only-unowned $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: $\mathcal{S}_{sb}'$ ts_{sb}' sb$'$ $\mathcal{O}_{sb}'$)
next
from unowned-shared-nth-update [OF i-bound ts_{sb}-i subset-refl]
show unowned-shared $\mathcal{S}_{sb}'$ ts_{sb}' **by** (simp add: ts_{sb}' $\mathcal{O}_{sb}'$ $\mathcal{S}_{sb}'$)
next

from no-outstanding-write-to-read-only-memory [OF i-bound ts_{sb}-i]
have no-write-to-read-only-memory $\mathcal{S}_{sb}$ sb.
hence no-write-to-read-only-memory $\mathcal{S}_{sb}$ (sb@[$Read_{sb}$ volatile a t (m_{sb} a)])
  by (simp add: no-write-to-read-only-memory-append)
from no-outstanding-write-to-read-only-memory-nth-update [OF i-bound this]
show no-outstanding-write-to-read-only-memory $\mathcal{S}_{sb}'$ ts_{sb}'
  by (simp add: ts_{sb}' $\mathcal{S}_{sb}'$ sb')
    qed

    have tmps-distinct': tmps-distinct ts_{sb}'
    proof (intro-locales)
from load-tmps-distinct [OF i-bound ts_{sb}-i]
have distinct-load-tmps is_{sb}'
  by (auto split: instr.splits simp add: is_{sb})
from load-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}')
    next
from read-tmps-distinct [OF i-bound ts_{sb}-i]
have distinct-read-tmps sb.
moreover
from load-tmps-read-tmps-distinct [OF i-bound ts_{sb}-i]
have t $\notin$ read-tmps sb
  by (auto simp add: is_{sb})
ultimately have distinct-read-tmps (sb @ [$Read_{sb}$ volatile a t (m_{sb} a)])
  by (auto simp add: distinct-read-tmps-append)
from read-tmps-distinct-nth-update [OF i-bound this]
show read-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}' sb')
    next
from load-tmps-read-tmps-distinct [OF i-bound ts_{sb}-i]
        load-tmps-distinct [OF i-bound ts_{sb}-i]
have load-tmps is_{sb}' $\cap$ read-tmps (sb @ [$Read_{sb}$ volatile a t (m_{sb} a)]) = {}
  by (clarsimp simp add: read-tmps-append is_{sb})
from load-tmps-read-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-read-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}' sb')
    qed

    have valid-sops': valid-sops ts_{sb}'
    proof −
from valid-store-sops [OF i-bound ts_{sb}-i]
have valid-store-sops': $\forall$sop$\in$store-sops is_{sb}'. valid-sop sop
  by (auto simp add: is_{sb})
from valid-write-sops [OF i-bound ts_{sb}-i]
have valid-write-sops': $\forall$sop$\in$write-sops (sb@ [$Read_{sb}$ volatile a t (m_{sb} a)]).
     valid-sop sop
  by (auto simp add: write-sops-append)
from valid-sops-nth-update [OF i-bound valid-write-sops' valid-store-sops']
show ?thesis **by** (simp add: ts_{sb}' sb')
    qed

    have valid-dd': valid-data-dependency ts_{sb}'

proof −
from data-dependency-consistent-instrs [OF i-bound ts_{sb}-i]
have dd-is: data-dependency-consistent-instrs (dom j_{sb}') is_{sb}'
by (auto simp add: is_{sb} j_{sb}')
from load-tmps-write-tmps-distinct [OF i-bound ts_{sb}-i]
have load-tmps $is_{sb}' \cap \bigcup$(fst ‘ write-sops (sb@ [$Read_{sb}$ volatile a t (m_{sb} a)])) = {}
by (auto simp add: write-sops-append is_{sb})
from valid-data-dependency-nth-update [OF i-bound dd-is this]
show ?thesis **by** (simp add: ts_{sb}' sb′)
qed

have load-tmps-fresh′: load-tmps-fresh ts_{sb}'
proof −
from load-tmps-fresh [OF i-bound ts_{sb}-i]
have load-tmps (Read volatile a t # is_{sb}') $\cap$ dom j_{sb} = {}
by (simp add: is_{sb})
moreover
from load-tmps-distinct [OF i-bound ts_{sb}-i] **have** t $\notin$ load-tmps is_{sb}'
by (auto simp add: is_{sb})
ultimately have load-tmps $is_{sb}' \cap$ dom (j_{sb}(t $\mapsto$ (m_{sb} a))) = {}
by auto
from load-tmps-fresh-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' sb′ j_{sb}')
qed

have enough-flushs′: enough-flushs ts_{sb}'
proof −
from clean-no-outstanding-volatile-$Write_{sb}$ [OF i-bound ts_{sb}-i]
have $\neg \mathcal{D}_{sb} \longrightarrow$ outstanding-refs is-volatile-$Write_{sb}$ (sb@[$Read_{sb}$ volatile a t (m_{sb} a)]) = {}
by (auto simp add: outstanding-refs-append)
from enough-flushs-nth-update [OF i-bound this]
show ?thesis
by (simp add: ts_{sb}' sb′ $\mathcal{D}_{sb}'$)
qed

have valid-program-history′: valid-program-history ts_{sb}'
proof −
from valid-program-history [OF i-bound ts_{sb}-i]
have causal-program-history is_{sb} sb .
then have causal′: causal-program-history is_{sb}' (sb@[$Read_{sb}$ volatile a t (m_{sb} a)])
by (auto simp: causal-program-history-Read is_{sb})
from valid-last-prog [OF i-bound ts_{sb}-i]
have last-prog p_{sb} sb = p_{sb}.
hence last-prog p_{sb} (sb @ [$Read_{sb}$ volatile a t (m_{sb} a)]) = p_{sb}
by (simp add: last-prog-append-$Read_{sb}$)
from valid-program-history-nth-update [OF i-bound causal′ this]
show ?thesis
by (simp add: ts_{sb}' sb′)
qed

show ?thesis
proof (cases outstanding-refs is-volatile-Write$_{\mathsf{sb}}$ sb = {})
case True

from True **have** flush-all: takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb = sb
by (auto simp add: outstanding-refs-conv)

from True **have** suspend-nothing: dropWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb = []
by (auto simp add: outstanding-refs-conv)

hence suspends-empty: suspends = []
by (simp add: suspends)
from suspends-empty is-sim **have** is: is = Read volatile a t # is$_{\mathsf{sb}}'$
by (simp add: is$_{\mathsf{sb}}$)
with suspends-empty ts-i
have ts-i: ts!i = (p$_{\mathsf{sb}}$, Read volatile a t # is$_{\mathsf{sb}}'$, j$_{\mathsf{sb}}$,(),
$\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{\mathsf{sb}}$, release ?take-sb (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}_{\mathsf{sb}}$)
by simp

from direct-memop-step.Read
have (Read volatile a t # is$_{\mathsf{sb}}'$,j$_{\mathsf{sb}}$, (), m,
$\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{\mathsf{sb}}$,release ?take-sb (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}_{\mathsf{sb}}$,$\mathcal{S}$) →
(is$_{\mathsf{sb}}'$, j$_{\mathsf{sb}}$(t ↦ m a), (), m, $\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{\mathsf{sb}}$,
release ?take-sb (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}_{\mathsf{sb}}$, $\mathcal{S}$)**.**
from direct-computation.concurrent-step.Memop [OF i-bound′ ts-i this]
have (ts, m, $\mathcal{S}$) $\Rightarrow_{\mathsf{d}}$ (ts[i := (p$_{\mathsf{sb}}$, is$_{\mathsf{sb}}'$, j$_{\mathsf{sb}}$(t ↦ m a), (),
$\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{\mathsf{sb}}$,release ?take-sb (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}_{\mathsf{sb}}$)], m, $\mathcal{S}$)**.**

moreover

from flush-all-until-volatile-write-Read-commute [OF i-bound ts$_{\mathsf{sb}}$-i [simplified is$_{\mathsf{sb}}$]]
have flush-commute: flush-all-until-volatile-write
(ts$_{\mathsf{sb}}$[i := (p$_{\mathsf{sb}}$,is$_{\mathsf{sb}}'$, j$_{\mathsf{sb}}$(t↦m$_{\mathsf{sb}}$ a), sb @ [Read$_{\mathsf{sb}}$ volatile a t (m$_{\mathsf{sb}}$ a)], $\mathcal{D}_{\mathsf{sb}}$, $\mathcal{O}_{\mathsf{sb}}$,$\mathcal{R}_{\mathsf{sb}}$)])

m$_{\mathsf{sb}}$ =
flush-all-until-volatile-write ts$_{\mathsf{sb}}$ m$_{\mathsf{sb}}$**.**

have flush-all-until-volatile-write ts$_{\mathsf{sb}}$ m$_{\mathsf{sb}}$ a = m$_{\mathsf{sb}}$ a
proof –
have ∀ j < length ts$_{\mathsf{sb}}$. i ≠ j ⟶
(let (-,-,-,sb$_{\mathsf{j}}$,-,-,-) = ts$_{\mathsf{sb}}$!j
in a ∉ outstanding-refs is-non-volatile-Write$_{\mathsf{sb}}$ (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb$_{\mathsf{j}}$))
proof –
{
fix j p$_{\mathsf{j}}$ is$_{\mathsf{j}}$ $\mathcal{O}_{\mathsf{j}}$ $\mathcal{R}_{\mathsf{j}}$ $\mathcal{D}_{\mathsf{j}}$ acq$_{\mathsf{j}}$ xs$_{\mathsf{j}}$ sb$_{\mathsf{j}}$
assume j-bound: j < length ts$_{\mathsf{sb}}$
assume neq-i-j: i ≠ j
assume jth: ts$_{\mathsf{sb}}$!j = (p$_{\mathsf{j}}$,is$_{\mathsf{j}}$, xs$_{\mathsf{j}}$, sb$_{\mathsf{j}}$, $\mathcal{D}_{\mathsf{j}}$, $\mathcal{O}_{\mathsf{j}}$, $\mathcal{R}_{\mathsf{j}}$)

have a $\notin$ outstanding-refs is-non-volatile-Write$_{sb}$ (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_j$)
proof
let ?take-sb$_j$ = (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_j$)
let ?drop-sb$_j$ = (dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_j$)
assume a-in: a $\in$ outstanding-refs is-non-volatile-Write$_{sb}$?take-sb$_j$
with outstanding-refs-takeWhile [**where** P$'$= Not $\circ$ is-volatile-Write$_{sb}$]
have a-in$'$: a $\in$ outstanding-refs is-non-volatile-Write$_{sb}$ sb$_j$
by auto
with non-volatile-owned-or-read-only-outstanding-non-volatile-writes
[OF outstanding-non-volatile-refs-owned-or-read-only [OF j-bound jth]]
have j-owns: a $\in$ $\mathcal{O}_j$ $\cup$ all-acquired sb$_j$
by auto
with ownership-distinct [OF i-bound j-bound neq-i-j ts$_{sb}$-i jth]
have a-not-owns: a $\notin$ $\mathcal{O}_{sb}$ $\cup$ all-acquired sb
by blast

from non-volatile-owned-or-read-only-append [of False $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb$_j$?drop-sb$_j$]
outstanding-non-volatile-refs-owned-or-read-only [OF j-bound jth]
have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb$_j$
by simp
from non-volatile-owned-or-read-only-outstanding-non-volatile-writes [OF this] a-in
have j-owns-drop: a $\in$ $\mathcal{O}_j$ $\cup$ all-acquired ?take-sb$_j$
by auto

from rels-cond [rule-format, OF j-bound [simplified leq] neq-i-j] ts-sim [rule-format, OF j-bound] jth
have no-unsharing:release ?take-sb$_j$ (dom ($\mathcal{S}_{sb}$)) $\mathcal{R}_j$ a $\neq$ Some False
by (auto simp add: Let-def)
{
assume a $\in$ acquired True sb $\mathcal{O}_{sb}$
with acquired-all-acquired-in [OF this] ownership-distinct [OF i-bound j-bound neq-i-j ts$_{sb}$-i jth]
j-owns
have False
by auto
}
moreover
{
assume a-ro: a $\in$ read-only (share ?drop-sb $\mathcal{S}$)
from read-only-share-unowned-in [OF weak-consis-drop a-ro] a-not-owns
acquired-all-acquired [of True ?take-sb $\mathcal{O}_{sb}$]
all-acquired-append [of ?take-sb ?drop-sb]
have a $\in$ read-only $\mathcal{S}$
by auto
with share-all-until-volatile-write-thread-local [OF ownership-distinct-ts$_{sb}$ sharing-consis-ts$_{sb}$ j-bound jth j-owns]
have a $\in$ read-only (share ?take-sb$_j$ $\mathcal{S}_{sb}$)
by (auto simp add: read-only-def $\mathcal{S}$)
hence a-dom: a $\in$ dom (share ?take-sb$_j$ $\mathcal{S}_{sb}$)

by (auto simp add: read-only-def domIff)
from outstanding-non-volatile-writes-unshared [OF j-bound jth]
non-volatile-writes-unshared-append [of $\mathcal{S}_{\mathsf{sb}}$?take-sb$_j$?drop-sb$_j$]
have nvw: non-volatile-writes-unshared $\mathcal{S}_{\mathsf{sb}}$?take-sb$_j$ **by** auto
from release-not-unshared-no-write-take [OF this no-unsharing a-dom] a-in
have False **by** auto
}
moreover
{
assume a-share: volatile $\wedge$ a $\in$ dom (share ?drop-sb $\mathcal{S}$)
from outstanding-non-volatile-writes-unshared [OF j-bound jth]
have non-volatile-writes-unshared $\mathcal{S}_{\mathsf{sb}}$ sb$_j$**.**
with non-volatile-writes-unshared-append [of $\mathcal{S}_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_j$)
(dropWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_j$)]
have unshared-take: non-volatile-writes-unshared $\mathcal{S}_{\mathsf{sb}}$ (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_j$)
by clarsimp

from valid-own **have** own-dist: ownership-distinct ts$_{\mathsf{sb}}$
by (simp add: valid-ownership-def)
from valid-sharing **have** sharing-consis $\mathcal{S}_{\mathsf{sb}}$ ts$_{\mathsf{sb}}$
by (simp add: valid-sharing-def)
from sharing-consistent-share-all-until-volatile-write [OF own-dist this i-bound ts$_{\mathsf{sb}}$-i]
have sc: sharing-consistent $\mathcal{S}$ (acquired True ?take-sb $\mathcal{O}_{\mathsf{sb}}$) ?drop-sb
by (simp add: $\mathcal{S}$)
from sharing-consistent-share-all-shared
have dom (share ?drop-sb $\mathcal{S}$) $\subseteq$ dom $\mathcal{S}$ $\cup$ all-shared ?drop-sb
by auto
also from sharing-consistent-all-shared [OF sc]
have ... $\subseteq$ dom $\mathcal{S}$ $\cup$ acquired True ?take-sb $\mathcal{O}_{\mathsf{sb}}$ **by** auto
also from acquired-all-acquired all-acquired-takeWhile
have ... $\subseteq$ dom $\mathcal{S}$ $\cup$ ($\mathcal{O}_{\mathsf{sb}}$ $\cup$ all-acquired sb) **by** force
finally
have a-shared: a $\in$ dom $\mathcal{S}$
using a-share a-not-owns
by auto

with share-all-until-volatile-write-thread-local [OF ownership-distinct-ts$_{\mathsf{sb}}$ sharing-consis-ts$_{\mathsf{sb}}$ j-bound jth j-owns]
have a-dom: a $\in$ dom (share ?take-sb$_j$ $\mathcal{S}_{\mathsf{sb}}$)
by (auto simp add: $\mathcal{S}$ domIff)
from release-not-unshared-no-write-take [OF unshared-take no-unsharing a-dom] a-in
have False **by** auto
}
ultimately show False
using access-cond$'$
by auto
qed

}
thus ?thesis
by (fastforce simp add: Let-def)
qed

from flush-all-until-volatile-write-buffered-val-conv
[OF True i-bound ts_{sb}-i this]
show ?thesis
by (simp add: buf-None)
qed

hence m-a: m a = m_{sb} a
by (simp add: m)

have tmps-commute: $j_{sb}(t \mapsto (m_{sb}\ a)) =$
$(j_{sb}$ |' (dom j_{sb} − {t}))(t $\mapsto$ (m_{sb} a))
apply (rule ext)
apply (auto simp add: restrict-map-def domIff)
done

from suspend-nothing
have suspend-nothing′: (dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb′) = []
by (simp add: sb′)

from $\mathcal{D}$
have $\mathcal{D}'$: $\mathcal{D}_{sb}$ = ($\mathcal{D}$ ∨ outstanding-refs is-volatile-$Write_{sb}$ (sb@[$Read_{sb}$ volatile a t (m_{sb} a)]) ≠ {})
by (auto simp: outstanding-refs-append)

have (ts_{sb}′,m_{sb},$\mathcal{S}_{sb}$′) ∼ (ts[i := (p_{sb},is_{sb}′, j_{sb}(t↦m a),(), $\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$,release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)], m,$\mathcal{S}$)
apply (rule sim-config.intros)
apply (simp add: m flush-commute ts_{sb}′ $\mathcal{O}_{sb}$′ $\mathcal{R}_{sb}$′ j_{sb}′ sb′ $\mathcal{D}_{sb}$′)
using share-all-until-volatile-write-Read-commute [OF i-bound ts_{sb}-i [simplified is_{sb}]]
apply (simp add: $\mathcal{S}$ $\mathcal{S}_{sb}$′ ts_{sb}′ sb′ $\mathcal{O}_{sb}$′ $\mathcal{R}_{sb}$′ j_{sb}′)
using leq
apply (simp add: ts_{sb}′)
using i-bound i-bound′ ts-sim ts-i True $\mathcal{D}'$
apply (clarsimp simp add: Let-def nth-list-update
outstanding-refs-conv m-a ts_{sb}′ $\mathcal{O}_{sb}$′ $\mathcal{R}_{sb}$′ $\mathcal{S}_{sb}$′ j_{sb}′ sb′ $\mathcal{D}_{sb}$′ suspend-nothing′
flush-all acquired-append release-append
split: if-split-asm)
apply (rule tmps-commute)
done

ultimately show ?thesis
using valid-own′ valid-hist′ valid-reads′ valid-sharing′ tmps-distinct′
valid-sops′ valid-dd′ load-tmps-fresh′ enough-flushs′
valid-program-history′ valid′
m_{sb}′ $\mathcal{S}_{sb}$′

by (auto simp del: fun-upd-apply)
next
case False

then obtain r **where** r-in: $r \in$ set sb **and** volatile-r: is-volatile-Write$_{sb}$ r
by (auto simp add: outstanding-refs-conv)
from takeWhile-dropWhile-real-prefix
[OF r-in, of (Not $\circ$ is-volatile-Write$_{sb}$), simplified, OF volatile-r]
obtain a′ v′ sb″ sop′ A′ L′ R′ W′ **where**
sb-split: sb = takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb @ Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′# sb″
and
drop: dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb = Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′# sb″
apply (auto)
subgoal for y ys
apply (case-tac y)
apply auto
done
done
from drop suspends **have** suspends: suspends = Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′# sb″
by simp

have (ts, m, $\mathcal{S}$) $\Rightarrow_{d}^{*}$ (ts, m, $\mathcal{S}$) **by** auto

moreover

note flush-commute = flush-all-until-volatile-write-Read-commute [OF i-bound ts$_{sb}$-i [simplified is$_{sb}$]]

have Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′$\in$ set sb
by (subst sb-split) auto

from dropWhile-append1 [OF this, of (Not $\circ$ is-volatile-Write$_{sb}$)]
have drop-app-comm:
(dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) (sb @ [Read$_{sb}$ volatile a t (m$_{sb}$ a)])) =
dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb @ [Read$_{sb}$ volatile a t (m$_{sb}$ a)]
by simp

from load-tmps-fresh [OF i-bound ts$_{sb}$-i]
have $t \notin$ dom j$_{sb}$
by (auto simp add: is$_{sb}$)
then have tmps-commute:
j$_{sb}$ |‘ (dom j$_{sb}$ − read-tmps sb″) =
j$_{sb}$ |‘ (dom j$_{sb}$ − insert t (read-tmps sb″))
apply −
apply (rule ext)
apply auto

done

from $\mathcal{D}$
have $\mathcal{D}'$: $\mathcal{D}_{\mathsf{sb}}$ = ($\mathcal{D}$ $\vee$ outstanding-refs is-volatile-Write$_{\mathsf{sb}}$ (sb@[Read$_{\mathsf{sb}}$ volatile a t (m$_{\mathsf{sb}}$ a)]) $\neq$ {})
by (auto simp: outstanding-refs-append)

have (ts$_{\mathsf{sb}}$',m$_{\mathsf{sb}}$,$\mathcal{S}_{\mathsf{sb}}$) $\sim$ (ts,m,$\mathcal{S}$)
apply (rule sim-config.intros)
apply (simp add: m flush-commute ts$_{\mathsf{sb}}$' $\mathcal{O}_{\mathsf{sb}}$' $\mathcal{R}_{\mathsf{sb}}$' j$_{\mathsf{sb}}$' sb')
using share-all-until-volatile-write-Read-commute [OF i-bound ts$_{\mathsf{sb}}$-i [simplified is$_{\mathsf{sb}}$]]
apply (simp add: $\mathcal{S}$ $\mathcal{S}_{\mathsf{sb}}$' ts$_{\mathsf{sb}}$' sb' $\mathcal{O}_{\mathsf{sb}}$' $\mathcal{R}_{\mathsf{sb}}$' j$_{\mathsf{sb}}$')
using leq
apply (simp add: ts$_{\mathsf{sb}}$')
using i-bound i-bound' ts-sim ts-i is-sim $\mathcal{D}'$
apply (clarsimp simp add: Let-def nth-list-update is-sim drop-app-comm
read-tmps-append suspends prog-instrs-append-Read$_{\mathsf{sb}}$ instrs-append-Read$_{\mathsf{sb}}$
hd-prog-append-Read$_{\mathsf{sb}}$
drop is$_{\mathsf{sb}}$ ts$_{\mathsf{sb}}$' sb' $\mathcal{O}_{\mathsf{sb}}$' $\mathcal{R}_{\mathsf{sb}}$' j$_{\mathsf{sb}}$' $\mathcal{D}_{\mathsf{sb}}$' acquired-append takeWhile-append1 [OF r-in] volatile-r split: if-split-asm)
apply (simp add: drop tmps-commute)+
done

ultimately show ?thesis
using valid-own' valid-hist' valid-reads' valid-sharing' tmps-distinct' valid-dd'
valid-sops' load-tmps-fresh' enough-flushs'
valid-program-history' valid'
m$_{\mathsf{sb}}$' $\mathcal{S}_{\mathsf{sb}}$'
by (auto simp del: fun-upd-apply)
qed
next
case (SBHWriteNonVolatile a D f A L R W)
then obtain
is$_{\mathsf{sb}}$: is$_{\mathsf{sb}}$ = Write False a (D, f) A L R W# is$_{\mathsf{sb}}$' **and**
$\mathcal{O}_{\mathsf{sb}}$': $\mathcal{O}_{\mathsf{sb}}$'=$\mathcal{O}_{\mathsf{sb}}$ **and**
$\mathcal{R}_{\mathsf{sb}}$': $\mathcal{R}_{\mathsf{sb}}$'=$\mathcal{R}_{\mathsf{sb}}$ **and**
j$_{\mathsf{sb}}$': j$_{\mathsf{sb}}$' = j$_{\mathsf{sb}}$ **and**
$\mathcal{D}_{\mathsf{sb}}$': $\mathcal{D}_{\mathsf{sb}}$'=$\mathcal{D}_{\mathsf{sb}}$ **and**
sb': sb'=sb@[Write$_{\mathsf{sb}}$ False a (D, f) (f j$_{\mathsf{sb}}$) A L R W] **and**
m$_{\mathsf{sb}}$': m$_{\mathsf{sb}}$' = m$_{\mathsf{sb}}$ **and**
$\mathcal{S}_{\mathsf{sb}}$': $\mathcal{S}_{\mathsf{sb}}$'=$\mathcal{S}_{\mathsf{sb}}$
by auto

from data-dependency-consistent-instrs [OF i-bound ts$_{\mathsf{sb}}$-i]
have D-tmps: D $\subseteq$ dom j$_{\mathsf{sb}}$
by (simp add: is$_{\mathsf{sb}}$)

from safe-memop-flush-sb [simplified is$_{\mathsf{sb}}$]

obtain a-owned$'$: a $\in$ acquired True sb $\mathcal{O}_{\mathsf{sb}}$ **and** a-unshared$'$: a $\notin$ dom (share ?drop-sb $\mathcal{S}$) **and**
rels-cond: $\forall$ j < length ts. i$\neq$j $\longrightarrow$ a $\notin$ dom (released (ts!j))

by cases auto

from a-owned$'$ acquired-all-acquired
have a-owned$''$: a $\in$ $\mathcal{O}_{\mathsf{sb}}$ $\cup$ all-acquired sb
by auto

{
fix j
fix p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j
assume j-bound: j < length ts_{sb}
assume ts_{sb}-j: ts_{sb}!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
assume neq-i-j: i $\neq$ j
have a $\notin$ $\mathcal{O}_j$ $\cup$ all-acquired sb_j
proof –
from ownership-distinct [OF i-bound j-bound neq-i-j ts_{sb}-i ts_{sb}-j] a-owned$''$
show ?thesis
by auto
qed
} **note** a-unowned-others = this

have a-unshared: a $\notin$ dom (share sb $\mathcal{S}_{\mathsf{sb}}$)
proof
assume a-share: a $\in$ dom (share sb $\mathcal{S}_{\mathsf{sb}}$)
from valid-sharing **have** sharing-consis $\mathcal{S}_{\mathsf{sb}}$ ts_{sb}
by (simp add: valid-sharing-def)
from in-shared-sb-share-all-until-volatile-write [OF this i-bound ts_{sb}-i a-owned$''$ a-share]
have a $\in$ dom (share ?drop-sb $\mathcal{S}$)
by (simp add: $\mathcal{S}$)
with a-unshared$'$
show False
by auto
qed

have valid-own$'$: valid-ownership $\mathcal{S}_{\mathsf{sb}}'$ ts_{sb}'
proof (intro-locales)
show outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}_{\mathsf{sb}}'$ ts_{sb}'
proof –
from outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ts_{sb}-i]
have non-volatile-owned-or-read-only False $\mathcal{S}_{\mathsf{sb}}$ $\mathcal{O}_{\mathsf{sb}}$ sb**.**

with a-owned$'$

have non-volatile-owned-or-read-only False $\mathcal{S}_{\mathsf{sb}}$ $\mathcal{O}_{\mathsf{sb}}$ (sb @ [Write$_{\mathsf{sb}}$ False a (D,f) (f j_{sb}) A L R W])
by (simp add: non-volatile-owned-or-read-only-append)
from outstanding-non-volatile-refs-owned-or-read-only-nth-update [OF i-bound this]
show ?thesis by (simp add: ts$_{\mathsf{sb}}$′ is$_{\mathsf{sb}}$ sb′ $\mathcal{O}_{\mathsf{sb}}$′ $\mathcal{S}_{\mathsf{sb}}$′)
qed
next
show outstanding-volatile-writes-unowned-by-others ts$_{\mathsf{sb}}$′
proof −
have outstanding-refs is-volatile-Write$_{\mathsf{sb}}$ (sb @ [Write$_{\mathsf{sb}}$ False a (D,f) (f j_{sb}) A L R W]) ⊆
outstanding-refs is-volatile-Write$_{\mathsf{sb}}$ sb
by (auto simp add: outstanding-refs-append)
from outstanding-volatile-writes-unowned-by-others-store-buffer
[OF i-bound ts$_{\mathsf{sb}}$-i this]
show ?thesis by (simp add: ts$_{\mathsf{sb}}$′ is$_{\mathsf{sb}}$ sb′ $\mathcal{O}_{\mathsf{sb}}$′ all-acquired-append)
qed
next
show read-only-reads-unowned ts$_{\mathsf{sb}}$′
proof −
have r: read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$)
(sb @ [Write$_{\mathsf{sb}}$ False a (D,f) (f j_{sb}) A L R W])) $\mathcal{O}_{\mathsf{sb}}$)
(dropWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) (sb @ [Write$_{\mathsf{sb}}$ False a (D,f) (f j_{sb}) A L R W]))
⊆
read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb) $\mathcal{O}_{\mathsf{sb}}$)
(dropWhile (Not ∘ is-volatile-Write$_{\mathsf{sb}}$) sb)
apply (case-tac outstanding-refs (is-volatile-Write$_{\mathsf{sb}}$) sb = {})
apply (simp-all add: outstanding-vol-write-take-drop-appends
acquired-append read-only-reads-append)
done
have $\mathcal{O}_{\mathsf{sb}}$ ∪ all-acquired (sb @ [Write$_{\mathsf{sb}}$ False a (D,f) (f j_{sb}) A L R W]) ⊆ $\mathcal{O}_{\mathsf{sb}}$ ∪ all-acquired sb
by (simp add: all-acquired-append)

from read-only-reads-unowned-nth-update [OF i-bound ts$_{\mathsf{sb}}$-i r this]
show ?thesis
by (simp add: ts$_{\mathsf{sb}}$′ $\mathcal{O}_{\mathsf{sb}}$′ sb′)
qed
next
show ownership-distinct ts$_{\mathsf{sb}}$′
proof −
from ownership-distinct-instructions-read-value-store-buffer-independent
[OF i-bound ts$_{\mathsf{sb}}$-i]
show ?thesis by (simp add: ts$_{\mathsf{sb}}$′ is$_{\mathsf{sb}}$ sb′ $\mathcal{O}_{\mathsf{sb}}$′ all-acquired-append)
qed
qed

have valid-hist′: valid-history program-step ts$_{\mathsf{sb}}$′

proof −
from valid-history [OF i-bound ts_{sb}-i]
have history-consistent j_{sb} (hd-prog p_{sb} sb) sb.
with valid-write-sops [OF i-bound ts_{sb}-i] D-tmps
valid-implies-valid-prog-hd [OF i-bound ts_{sb}-i valid]
have history-consistent j_{sb} (hd-prog p_{sb} (sb@[$Write_{sb}$ False a (D,f) (f j_{sb}) A L R W]))
(sb@ [$Write_{sb}$ False a (D,f) (f j_{sb}) A L R W])
apply −
apply (rule history-consistent-appendI)
apply (auto simp add: hd-prog-append-$Write_{sb}$)
done
from valid-history-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' is_{sb} sb' $\mathcal{O}_{sb}'$ j_{sb}')
qed

have valid-reads': valid-reads m_{sb} ts_{sb}'
proof −
from valid-reads [OF i-bound ts_{sb}-i]
have reads-consistent False $\mathcal{O}_{sb}$ m_{sb} sb .
from reads-consistent-snoc-$Write_{sb}$ [OF this]
have reads-consistent False $\mathcal{O}_{sb}$ m_{sb} (sb @ [$Write_{sb}$ False a (D,f) (f j_{sb}) A L R W]).
from valid-reads-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' is_{sb} sb' $\mathcal{O}_{sb}'$ j_{sb}')
qed

have valid-sharing': valid-sharing $\mathcal{S}_{sb}'$ ts_{sb}'
proof (intro-locales)
from outstanding-non-volatile-writes-unshared [OF i-bound ts_{sb}-i] a-unshared
have non-volatile-writes-unshared $\mathcal{S}_{sb}$
(sb @ [$Write_{sb}$ False a (D,f) (f j_{sb}) A L R W])
by (auto simp add: non-volatile-writes-unshared-append)
from outstanding-non-volatile-writes-unshared-nth-update [OF i-bound this]
show outstanding-non-volatile-writes-unshared $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: ts_{sb}' is_{sb} sb' $\mathcal{O}_{sb}'$ j_{sb}' $\mathcal{S}_{sb}'$)
next
from sharing-consis [OF i-bound ts_{sb}-i]
have sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ sb.
then
have sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ (sb @ [$Write_{sb}$ False a (D,f) (f j_{sb}) A L R W])
by (simp add: sharing-consistent-append)
from sharing-consis-nth-update [OF i-bound this]
show sharing-consis $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: ts_{sb}' $\mathcal{O}_{sb}'$ sb' $\mathcal{S}_{sb}'$)
next
from read-only-unowned-nth-update [OF i-bound read-only-unowned [OF i-bound ts_{sb}-i]
]
show read-only-unowned $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: $\mathcal{S}_{sb}'$ ts_{sb}' $\mathcal{O}_{sb}'$)
next
from unowned-shared-nth-update [OF i-bound ts_{sb}-i subset-refl]

show unowned-shared $\mathcal{S}_{\mathsf{sb}}'$ $\mathrm{ts}_{\mathsf{sb}}'$
  by (simp add: $\mathrm{ts}_{\mathsf{sb}}'$ $\mathrm{is}_{\mathsf{sb}}$ sb′ $\mathcal{O}_{\mathsf{sb}}'$ $\mathrm{j}_{\mathsf{sb}}'$ $\mathcal{S}_{\mathsf{sb}}'$)
    next
from a-unshared
have a $\notin$ read-only (share sb $\mathcal{S}_{\mathsf{sb}}$)
  by (auto simp add: read-only-def dom-def)
with no-outstanding-write-to-read-only-memory [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i]

have no-write-to-read-only-memory $\mathcal{S}_{\mathsf{sb}}$ (sb @ [$\mathrm{Write}_{\mathsf{sb}}$ False a (D,f) (f j_{sb}) A L R W])
  by (simp add: no-write-to-read-only-memory-append)

from no-outstanding-write-to-read-only-memory-nth-update [OF i-bound this]
show no-outstanding-write-to-read-only-memory $\mathcal{S}_{\mathsf{sb}}'$ $\mathrm{ts}_{\mathsf{sb}}'$
  by (simp add: $\mathcal{S}_{\mathsf{sb}}'$ $\mathrm{ts}_{\mathsf{sb}}'$ sb′)
    qed

    have tmps-distinct′: tmps-distinct $\mathrm{ts}_{\mathsf{sb}}'$
    proof (intro-locales)
from load-tmps-distinct [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i]
have distinct-load-tmps $\mathrm{is}_{\mathsf{sb}}'$
  by (auto split: instr.splits simp add: $\mathrm{is}_{\mathsf{sb}}$)
from load-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-distinct $\mathrm{ts}_{\mathsf{sb}}'$
  by (simp add: $\mathrm{ts}_{\mathsf{sb}}'$ $\mathrm{is}_{\mathsf{sb}}$ sb′ $\mathcal{O}_{\mathsf{sb}}'$ $\mathrm{j}_{\mathsf{sb}}'$)
    next
from read-tmps-distinct [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i]
have distinct-read-tmps sb**.**
hence distinct-read-tmps (sb @ [$\mathrm{Write}_{\mathsf{sb}}$ False a (D,f) (f j_{sb}) A L R W])
  by (simp add: distinct-read-tmps-append)
from read-tmps-distinct-nth-update [OF i-bound this]
show read-tmps-distinct $\mathrm{ts}_{\mathsf{sb}}'$
  by (simp add: $\mathrm{ts}_{\mathsf{sb}}'$ $\mathrm{is}_{\mathsf{sb}}$ sb′ $\mathcal{O}_{\mathsf{sb}}'$ $\mathrm{j}_{\mathsf{sb}}'$)
    next
from load-tmps-read-tmps-distinct [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i]
      load-tmps-distinct [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i]
have load-tmps $\mathrm{is}_{\mathsf{sb}}'$ $\cap$ read-tmps (sb @ [$\mathrm{Write}_{\mathsf{sb}}$ False a (D,f) (f j_{sb}) A L R W]) = {}
  by (clarsimp simp add: read-tmps-append $\mathrm{is}_{\mathsf{sb}}$)
from load-tmps-read-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-read-tmps-distinct $\mathrm{ts}_{\mathsf{sb}}'$
  by (simp add: $\mathrm{ts}_{\mathsf{sb}}'$ $\mathrm{is}_{\mathsf{sb}}$ sb′ $\mathcal{O}_{\mathsf{sb}}'$ $\mathrm{j}_{\mathsf{sb}}'$)
    qed

    have valid-sops′: valid-sops $\mathrm{ts}_{\mathsf{sb}}'$
    proof –
from valid-store-sops [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i]
obtain valid-Df: valid-sop (D,f) **and**
  valid-store-sops′: $\forall$sop$\in$store-sops $\mathrm{is}_{\mathsf{sb}}'$. valid-sop sop
  by (auto simp add: $\mathrm{is}_{\mathsf{sb}}$)
from valid-Df valid-write-sops [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i]
have valid-write-sops′: $\forall$sop$\in$write-sops (sb@ [$\mathrm{Write}_{\mathsf{sb}}$ False a (D, f) (f j_{sb}) A L R W]).

valid-sop sop
by (auto simp add: write-sops-append)
from valid-sops-nth-update [OF i-bound valid-write-sops′ valid-store-sops′]
show ?thesis
by (simp add: ts_{sb}′ is_{sb} sb′ $\mathcal{O}_{sb}$′ j_{sb}′)
qed

have valid-dd′: valid-data-dependency ts_{sb}′
proof −
from data-dependency-consistent-instrs [OF i-bound ts_{sb}-i]
obtain D-indep: D ∩ load-tmps is_{sb}′ = {} **and**
dd-is: data-dependency-consistent-instrs (dom j_{sb}′) is_{sb}′
by (auto simp add: is_{sb} j_{sb}′)
from load-tmps-write-tmps-distinct [OF i-bound ts_{sb}-i] D-indep
have load-tmps is_{sb}′ ∩
⋃(fst ‘ write-sops (sb@ [$Write_{sb}$ False a (D, f) (f j_{sb}) A L R W])) = {}
by (auto simp add: write-sops-append is_{sb})
from valid-data-dependency-nth-update [OF i-bound dd-is this]
show ?thesis
by (simp add: ts_{sb}′ is_{sb} sb′ $\mathcal{O}_{sb}$′ j_{sb}′)
qed

have load-tmps-fresh′: load-tmps-fresh ts_{sb}′
proof −
from load-tmps-fresh [OF i-bound ts_{sb}-i]
have load-tmps is_{sb}′ ∩ dom j_{sb} = {}
by (auto simp add: is_{sb})
from load-tmps-fresh-nth-update [OF i-bound this]
show ?thesis
by (simp add: ts_{sb}′ is_{sb} sb′ $\mathcal{O}_{sb}$′ j_{sb}′)
qed

have enough-flushs′: enough-flushs ts_{sb}′
proof −
from clean-no-outstanding-volatile-$Write_{sb}$ [OF i-bound ts_{sb}-i]
have ¬ $\mathcal{D}_{sb}$ ⟶ outstanding-refs is-volatile-$Write_{sb}$ (sb@[$Write_{sb}$ False a (D,f) (f j_{sb}) A L R W]) = {}
by (auto simp add: outstanding-refs-append)
from enough-flushs-nth-update [OF i-bound this]
show ?thesis
by (simp add: ts_{sb}′ sb′ $\mathcal{D}_{sb}$′)
qed

have valid-program-history′: valid-program-history ts_{sb}′
proof −
from valid-program-history [OF i-bound ts_{sb}-i]
have causal-program-history is_{sb} sb **.**
then have causal′: causal-program-history is_{sb}′ (sb@[$Write_{sb}$ False a (D,f) (f j_{sb}) A L R W])

by (auto simp: causal-program-history-Write is_{sb})
from valid-last-prog [OF i-bound ts_{sb}-i]
have last-prog p_{sb} sb = p_{sb}.
hence last-prog p_{sb} (sb @ [$Write_{sb}$ False a (D,f) (f j_{sb}) A L R W]) = p_{sb}
by (simp add: last-prog-append-$Write_{sb}$)
from valid-program-history-nth-update [OF i-bound causal′ this]
show ?thesis
by (simp add: ts_{sb}′ sb′)
qed

from valid-store-sops [OF i-bound ts_{sb}-i, rule-format]
have valid-sop (D,f) **by** (auto simp add: is_{sb})
then interpret valid-sop (D,f) **.**

show ?thesis
proof (cases outstanding-refs is-volatile-$Write_{sb}$ sb = {})
case True

from True **have** flush-all: takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb = sb
by (auto simp add: outstanding-refs-conv)

from True **have** suspend-nothing: dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb = []
by (auto simp add: outstanding-refs-conv)

hence suspends-empty: suspends = []
by (simp add: suspends)

from suspends-empty is-sim **have** is: is = Write False a (D,f) A L R W# is_{sb}′
by (simp add: is_{sb})
with suspends-empty ts-i
have ts-i: ts!i = (p_{sb}, Write False a (D,f) A L R W# is_{sb}′,
j_{sb},(),
$\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$,release ?take-sb (dom ($\mathcal{S}_{sb}$)) $\mathcal{R}_{sb}$)
by simp

from direct-memop-step.WriteNonVolatile [OF]
have (Write False a (D, f) A L R W# is_{sb}′,
j_{sb}, (),m,$\mathcal{D}$,acquired True ?take-sb $\mathcal{O}_{sb}$,release ?take-sb (dom ($\mathcal{S}_{sb}$)) $\mathcal{R}_{sb}$, $\mathcal{S}$) →
(is_{sb}′,
j_{sb}, (), m(a := f j_{sb}), $\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$,
release ?take-sb (dom ($\mathcal{S}_{sb}$)) $\mathcal{R}_{sb}$, $\mathcal{S}$).
from direct-computation.concurrent-step.Memop [OF i-bound′ ts-i this]
have (ts, m, $\mathcal{S}$) $\Rightarrow_d$
(ts[i := (p_{sb}, is_{sb}′, j_{sb}, (),$\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$,
release ?take-sb (dom ($\mathcal{S}_{sb}$)) $\mathcal{R}_{sb}$)],
m(a := f j_{sb}),$\mathcal{S}$).

moreover

have $\forall$ j<length ts_{sb}. i $\neq$ j $\longrightarrow$
(let (-,-, -, sb_j,-,-,-) = ts_{sb} ! j
in a $\notin$ outstanding-refs is-non-volatile-$Write_{sb}$ (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j))
proof –
{
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ acq_j xs_j sb_j
assume j-bound: j < length ts_{sb}
assume neq-i-j: i $\neq$ j
assume jth: ts_{sb}!j = (p_j,is_j, xs_j, sb_j, $\mathcal{D}_j$, $\mathcal{O}_j$,$\mathcal{R}_j$)
have a $\notin$ outstanding-refs is-non-volatile-$Write_{sb}$ (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
proof
assume a-in: a $\in$ outstanding-refs is-non-volatile-$Write_{sb}$ (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
hence a $\in$ outstanding-refs is-non-volatile-$Write_{sb}$ sb_j
using outstanding-refs-append [of is-non-volatile-$Write_{sb}$ (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
(dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)]
by auto
with non-volatile-owned-or-read-only-outstanding-non-volatile-writes
[OF outstanding-non-volatile-refs-owned-or-read-only [OF j-bound jth]]
have j-owns: a $\in$ $\mathcal{O}_j$ $\cup$ all-acquired sb_j
by auto

from j-owns a-owned'' ownership-distinct [OF i-bound j-bound neq-i-j ts_{sb}-i jth]
show False
by auto
qed
}
thus ?thesis **by** (fastforce simp add: Let-def)
qed

note flush-commute = flush-all-until-volatile-write-append-non-volatile-write-commute
[OF True i-bound ts_{sb}-i this]

from suspend-nothing
have suspend-nothing': (dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb') = []
by (simp add: sb')

from $\mathcal{D}$
have $\mathcal{D}'$: $\mathcal{D}_{sb}$ = ($\mathcal{D}$ $\vee$ outstanding-refs is-volatile-$Write_{sb}$
(sb@[$Write_{sb}$ False a (D,f) (f j_{sb}) A L R W]) $\neq$ {})
by (auto simp: outstanding-refs-append)

have (ts_{sb}',m_{sb},$\mathcal{S}_{sb}$') $\sim$
(ts[i := (p_{sb},is_{sb}', j_{sb},(),$\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$,
release ?take-sb (dom ($\mathcal{S}_{sb}$)) $\mathcal{R}_{sb}$)],
m(a:=f j_{sb}),$\mathcal{S}$)

apply (rule sim-config.intros)
apply (simp add: m flush-commute ts_{sb}' $\mathcal{O}_{sb}'$ $\mathcal{R}_{sb}'$ sb' j_{sb}' $\mathcal{D}_{sb}'$)
using share-all-until-volatile-write-Write-commute
[OF i-bound ts_{sb}-i [simplified is_{sb}]]
apply (clarsimp simp add: $\mathcal{S}$ $\mathcal{S}_{sb}'$ ts_{sb}' sb' $\mathcal{O}_{sb}'$ $\mathcal{R}_{sb}'$ j_{sb}')
using leq
apply (simp add: ts_{sb}')
using i-bound i-bound' ts-sim ts-i True $\mathcal{D}'$
apply (clarsimp simp add: Let-def nth-list-update
outstanding-refs-conv ts_{sb}' $\mathcal{O}_{sb}'$ $\mathcal{R}_{sb}'$ $\mathcal{S}_{sb}'$ j_{sb}' sb' $\mathcal{D}_{sb}'$ suspend-nothing' flush-all
acquired-append release-append split: if-split-asm)
done

ultimately
show ?thesis
using valid-own' valid-hist' valid-reads' valid-sharing' tmps-distinct' valid-sops'
valid-dd' load-tmps-fresh' enough-flushs'
valid-program-history' valid' m_{sb}' $\mathcal{S}_{sb}'$
by (auto simp del: fun-upd-apply)
next

case False

then obtain r **where** r-in: r $\in$ set sb **and** volatile-r: is-volatile-Write_{sb} r
by (auto simp add: outstanding-refs-conv)
from takeWhile-dropWhile-real-prefix
[OF r-in, of (Not $\circ$ is-volatile-Write_{sb}), simplified, OF volatile-r]
obtain a' v' sb'' sop' A' L' R' W' **where**
sb-split: sb = takeWhile (Not $\circ$ is-volatile-Write_{sb}) sb @ Write_{sb} True a' sop' v' A' L' R' W'# sb''
and
drop: dropWhile (Not $\circ$ is-volatile-Write_{sb}) sb = Write_{sb} True a' sop' v' A' L' R' W'# sb''
apply (auto)
subgoal for y ys
apply (case-tac y)
apply auto
done
done
from drop suspends **have** suspends: suspends = Write_{sb} True a' sop' v' A' L' R' W'# sb''
by simp

have (ts, m, $\mathcal{S}$) $\Rightarrow_d{}^*$ (ts, m, $\mathcal{S}$) **by** auto

moreover

note flush-commute =
flush-all-until-volatile-write-append-unflushed [OF False i-bound ts_{sb}-i]

have Write_{sb} True a′ sop′ v′ A′ L′ R′ W′ ∈ set sb
by (subst sb-split) auto
note drop-app = dropWhile-append1 [OF this, of (Not ∘ is-volatile-Write_{sb}), simplified]

from $\mathcal{D}$
have $\mathcal{D}'$: $\mathcal{D}_{sb}$ = ($\mathcal{D}$ ∨ outstanding-refs is-volatile-Write_{sb} (sb@[Write_{sb} False a (D,f) (f j_{sb}) A L R W]) ≠ {})
by (auto simp: outstanding-refs-append)

have (ts_{sb}',m_{sb},$\mathcal{S}_{sb}'$) ∼ (ts,m,$\mathcal{S}$)
apply (rule sim-config.intros)
apply (simp add: m flush-commute ts_{sb}' $\mathcal{O}_{sb}'$ $\mathcal{R}_{sb}'$ j_{sb}' sb′)
using share-all-until-volatile-write-Write-commute
[OF i-bound ts_{sb}-i [simplified is_{sb}]]
apply (clarsimp simp add: $\mathcal{S}$ $\mathcal{S}_{sb}'$ ts_{sb}' sb′ $\mathcal{O}_{sb}'$ $\mathcal{R}_{sb}'$ j_{sb}')
using leq
apply (simp add: ts_{sb}')
using i-bound i-bound′ ts-sim ts-i is-sim $\mathcal{D}'$
apply (clarsimp simp add: Let-def nth-list-update is-sim drop-app
read-tmps-append suspends
prog-instrs-append-Write_{sb} instrs-append-Write_{sb} hd-prog-append-Write_{sb}
drop is_{sb} ts_{sb}' sb′ $\mathcal{O}_{sb}'$ $\mathcal{R}_{sb}'$ $\mathcal{S}_{sb}'$
j_{sb}' $\mathcal{D}_{sb}'$ acquired-append takeWhile-append1 [OF r-in]
volatile-r
split: if-split-asm)
done
ultimately show ?thesis
using valid-own′ valid-hist′ valid-reads′ valid-sharing′ tmps-distinct′ valid-dd′
valid-sops′ load-tmps-fresh′ enough-flushs′
valid-program-history′ valid′ m_{sb}' $\mathcal{S}_{sb}'$
by (auto simp del: fun-upd-apply)
qed
next
case (SBHWriteVolatile a D f A L R W)
then obtain
is_{sb}: is_{sb} = Write True a (D, f) A L R W# is_{sb}' **and**
$\mathcal{O}_{sb}'$: $\mathcal{O}_{sb}'$=$\mathcal{O}_{sb}$ **and**
$\mathcal{R}_{sb}'$: $\mathcal{R}_{sb}'$=$\mathcal{R}_{sb}$ **and**
j_{sb}': j_{sb}' = j_{sb} **and**
$\mathcal{D}_{sb}'$: $\mathcal{D}_{sb}'$=True **and**
sb′: sb′=sb@[Write_{sb} True a (D, f) (f j_{sb}) A L R W] **and**
m_{sb}': m_{sb}' = m_{sb} **and**
$\mathcal{S}_{sb}'$: $\mathcal{S}_{sb}'$=$\mathcal{S}_{sb}$
by auto

from data-dependency-consistent-instrs [OF i-bound ts_{sb}-i]
have D-subset: D ⊆ dom j_{sb}
by (simp add: is_{sb})

from safe-memop-flush-sb [simplified is_{sb}] **obtain**
a-unowned-others-ts:
$\forall$j<length (map owned ts). i $\neq$ j $\longrightarrow$ (a $\notin$ owned (ts!j) $\cup$ dom (released (ts!j)))
and
L-subset: L $\subseteq$ A **and**
A-shared-owned: A $\subseteq$ dom (share ?drop-sb $\mathcal{S}$) $\cup$ acquired True sb $\mathcal{O}_{\mathsf{sb}}$ **and**
R-acq: R $\subseteq$ acquired True sb $\mathcal{O}_{\mathsf{sb}}$ **and**
A-R: A $\cap$ R = {} **and**
A-unowned-by-others-ts:
$\forall$j<length (map owned ts). i$\neq$j $\longrightarrow$ (A $\cap$ (owned (ts!j) $\cup$ dom (released (ts!j))) = {})
and
a-not-ro$'$: a $\notin$ read-only (share ?drop-sb $\mathcal{S}$)
by cases auto

from a-unowned-others-ts ts-sim leq
have a-unowned-others:
$\forall$j<length ts_{sb}. i $\neq$ j $\longrightarrow$
(let (-,-,-,sb_{j},-,$\mathcal{O}_{\mathsf{j}}$,-) = ts_{sb}!j in
a $\notin$ acquired True (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{j}) $\mathcal{O}_{\mathsf{j}}$ $\wedge$
a $\notin$ all-shared (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{j}))
apply (clarsimp simp add: Let-def)
subgoal for j
apply (drule-tac x=j **in** spec)
apply (auto simp add: dom-release-takeWhile)
done
done

have a-not-ro: a $\notin$ read-only (share sb $\mathcal{S}_{\mathsf{sb}}$)
proof
assume a: a $\in$ read-only (share sb $\mathcal{S}_{\mathsf{sb}}$)
from local.read-only-unowned-axioms **have** read-only-unowned $\mathcal{S}_{\mathsf{sb}}$ ts_{sb}.
from in-read-only-share-all-until-volatile-write$'$ [OF ownership-distinct-ts_{sb} sharing-consis-ts_{sb}
‹read-only-unowned $\mathcal{S}_{\mathsf{sb}}$ ts_{sb}› i-bound ts_{sb}-i a-unowned-others a]
have a $\in$ read-only (share ?drop-sb $\mathcal{S}$)
by (simp add: $\mathcal{S}$)
with a-not-ro$'$ **show** False **by** simp
qed

from A-unowned-by-others-ts ts-sim leq
have A-unowned-by-others:
$\forall$j<length ts_{sb}. i$\neq$j $\longrightarrow$ (let (-,-,-,sb_{j},-,$\mathcal{O}_{\mathsf{j}}$,-) = ts_{sb}!j
in A $\cap$ (acquired True (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{j}) $\mathcal{O}_{\mathsf{j}}$ $\cup$
all-shared (takeWhile (Not $\circ$ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{j})) = {})
apply (clarsimp simp add: Let-def)
subgoal for j
apply (drule-tac x=j **in** spec)
apply (force simp add: dom-release-takeWhile)

done
done
have a-not-acquired-others: $\forall$j<length (map $\mathcal{O}$-sb ts_{sb}). i $\neq$ j $\longrightarrow$
(let ($\mathcal{O}_j$,sb_j) = (map $\mathcal{O}$-sb ts_{sb})!j in a $\notin$ all-acquired sb_j)
proof –
{
fix j $\mathcal{O}_j$ sb_j
assume j-bound: j < length (map owned ts_{sb})
assume neq-i-j: i$\neq$j
assume ts_{sb}-j: (map $\mathcal{O}$-sb ts_{sb})!j = ($\mathcal{O}_j$,sb_j)
assume conflict: a $\in$ all-acquired sb_j
have False
proof –
from j-bound leq
have j-bound′: j < length (map owned ts)
by auto
from j-bound **have** j-bound′′: j < length ts_{sb}
by auto
from j-bound′ **have** j-bound′′′: j < length ts
by simp

let ?take-sb_j = (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
let ?drop-sb_j = (dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)

from ts-sim [rule-format, OF j-bound′′] ts_{sb}-j j-bound′′

obtain p_j $suspends_j$ is_{sbj} $\mathcal{R}_j$ $\mathcal{D}_{sbj}$ $\mathcal{D}_j$ j_{sbj} is_j **where**
ts_{sb}-j: ts_{sb} ! j = (p_j,is_{sbj}, j_{sbj}, sb_j, $\mathcal{D}_{sbj}$,$\mathcal{O}_j$,$\mathcal{R}_j$) **and**
$suspends_j$: $suspends_j$ = dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j **and**
is_j: instrs $suspends_j$ @ is_{sbj} = is_j @ prog-instrs $suspends_j$ **and**
$\mathcal{D}_j$: $\mathcal{D}_{sbj}$ = ($\mathcal{D}_j$ $\vee$ outstanding-refs is-volatile-$Write_{sb}$ sb_j $\neq$ {}) **and**
ts_j: ts!j = (hd-prog p_j $suspends_j$, is_j,
j_{sbj} |‘ (dom j_{sbj} – read-tmps $suspends_j$),(),
$\mathcal{D}_j$,
acquired True ?take-sb_j $\mathcal{O}_j$,
release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
apply (cases ts_{sb}!j)
apply (force simp add: Let-def)
done

from a-unowned-others [rule-format,OF - neq-i-j] ts_{sb}-j j-bound
obtain a-unacq: a $\notin$ acquired True ?take-sb_j $\mathcal{O}_j$ **and** a-not-shared: a $\notin$ all-shared ?take-sb_j
by auto
have conflict-drop: a $\in$ all-acquired $suspends_j$
proof (rule ccontr)
assume a $\notin$ all-acquired $suspends_j$
with all-acquired-append [of ?take-sb_j ?drop-sb_j] conflict
have a $\in$ all-acquired ?take-sb_j

by (auto simp add: suspends_j)
from all-acquired-unshared-acquired [OF this a-not-shared] a-unacq
show False **by** auto
qed

from j-bound′′′ i-bound′ **have** j-bound-ts′: j < length ?ts′
by simp

from split-all-acquired-in [OF conflict-drop]
show ?thesis
proof
assume ∃sop a′ v ys zs A L R W.
suspends_j = ys @ Write_{sb} True a′ sop v A L R W# zs ∧ a ∈ A
then
obtain a′ sop′ v′ ys zs A′ L′ R′ W′ **where**
split-suspends_j: suspends_j = ys @ Write_{sb} True a′ sop′ v′ A′ L′ R′ W′# zs
(**is** suspends_j = ?suspends) **and**
a-A′: a ∈ A′
by blast

from sharing-consis [OF j-bound′′ ts_{sb}-j]
have sharing-consis-j: sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb_j**.**
then have A′-R′: A′ ∩ R′ = {}
by (simp add: sharing-consistent-append [of - - ?take-sb_j ?drop-sb_j, simplified]
suspends_j [symmetric] split-suspends_j sharing-consistent-append)
from valid-program-history [OF j-bound′′ ts_{sb}-j]
have causal-program-history is_{sbj} sb_j**.**
then have cph: causal-program-history is_{sbj} ?suspends
apply −
apply (rule causal-program-history-suffix [**where** sb=?take-sb_j])
apply (simp only: split-suspends_j [symmetric] suspends_j)
apply (simp add: split-suspends_j)
done

from ts_j neq-i-j j-bound
have ts′-j: ?ts′!j = (hd-prog p_j suspends_j, is_j,
j_{sbj} |‘ (dom j_{sbj} − read-tmps suspends_j),(),
$\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$,release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
by auto
from valid-last-prog [OF j-bound′′ ts_{sb}-j] **have** last-prog: last-prog p_j sb_j = p_j**.**
then
have lp: last-prog p_j suspends_j = p_j
apply −
apply (rule last-prog-same-append [**where** sb=?take-sb_j])
apply (simp only: split-suspends_j [symmetric] suspends_j)
apply simp
done

from valid-reads [OF j-bound″ ts_{sb}-j]
have reads-consis-j: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.

from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts_{sb}›
j-bound″ ts_{sb}-j this]
have reads-consis-m-j: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
by (simp add: m $suspends_j$)

from outstanding-non-write-non-vol-reads-drop-disj [OF i-bound j-bound″ neq-i-j ts_{sb}-i ts_{sb}-j]
have outstanding-refs is-$Write_{sb}$?drop-sb ∩ outstanding-refs is-non-volatile-$Read_{sb}$ $suspends_j$ = {}
by (simp add: $suspends_j$)
from reads-consistent-flush-independent [OF this reads-consis-m-j]
have reads-consis-flush-suspend: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) (flush ?drop-sb m) $suspends_j$.

hence reads-consis-ys: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) (flush ?drop-sb m) (ys@[$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′])
by (simp add: split-$suspends_j$ reads-consistent-append)

from valid-write-sops [OF j-bound″ ts_{sb}-j]
have ∀ sop∈write-sops (?take-sb_j@?suspends). valid-sop sop
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain valid-sops-take: ∀ sop∈write-sops ?take-sb_j. valid-sop sop **and**
valid-sops-drop: ∀ sop∈write-sops (ys@[$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]). valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound″ ts_{sb}-j]
have distinct-read-tmps (?take-sb_j@$suspends_j$)
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb_j ∩ read-tmps $suspends_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps $suspends_j$
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp only: distinct-read-tmps-append)
done

from valid-history [OF j-bound″ ts_{sb}-j]
have h-consis:
history-consistent j_{sb_j} (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof −
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop h-consis] last-prog-hd-prog
have hist-consis′: history-consistent j_{sbj} (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis-j]
have no-vol-read: outstanding-refs is-volatile-$Read_{sb}$
(ys@[$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]) = {}
by (auto simp add: outstanding-refs-append $suspends_j$ [symmetric]
split-$suspends_j$)

have acq-simp:
acquired True (ys @ [$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′])
(acquired True ?take-sb_j $\mathcal{O}_j$) =
acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$) ∪ A′ − R′
by (simp add: acquired-append)

from flush-store-buffer-append [**where** sb=ys@[$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]
and sb′=zs, simplified,
OF j-bound-ts′ is_j [simplified split-$suspends_j$] cph [simplified $suspends_j$]
ts′-j [simplified split-$suspends_j$] refl lp [simplified split-$suspends_j$] reads-consis-ys
hist-consis′ [simplified split-$suspends_j$] valid-sops-drop
distinct-read-tmps-drop [simplified split-$suspends_j$]
no-volatile-$Read_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=share ?drop-sb $\mathcal{S}$]
obtain is_j′ $\mathcal{R}_j$′ **where**
is_j′: instrs zs @ is_{sbj} = is_j′ @ prog-instrs zs **and**
steps-ys: (?ts′, flush ?drop-sb m, share ?drop-sb $\mathcal{S}$) $\Rightarrow_d{}^*$
(?ts′[j:=(last-prog
(hd-prog p_j ($Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′# zs)) (ys@[$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]),
is_j′,
j_{sbj} |‘ (dom j_{sbj} − read-tmps zs),

(), True, acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$) ∪ A′ − R′,$\mathcal{R}_j$′)],
flush (ys@[$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]) (flush ?drop-sb m),
share (ys@[$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]) (share ?drop-sb $\mathcal{S}$))
(**is** (-,-,-) $\Rightarrow_d^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto simp add: acquired-append outstanding-refs-append)

from i-bound′ **have** i-bound-ys: i < length ?ts-ys
by auto

from i-bound′ neq-i-j
have ts-ys-i: ?ts-ys!i = (p_{sb}, is_{sb}, j_{sb},(),
$\mathcal{D}_{sb}$, acquired True sb $\mathcal{O}_{sb}$, release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
by simp
note conflict-computation = rtranclp-trans [OF steps-flush-sb steps-ys]

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys).

from safe-delayedE [OF this i-bound-ys ts-ys-i, simplified is_{sb}]
have a-unowned:
∀j < length ?ts-ys. i≠j ⟶ (let ($\mathcal{O}_j$) = map owned ?ts-ys!j in a ∉ $\mathcal{O}_j$)
apply cases
apply (auto simp add: Let-def is_{sb})
done
from a-A′ a-unowned [rule-format, of j] neq-i-j j-bound′ A′-R′
show False
by (auto simp add: Let-def)
next
assume ∃A L R W ys zs. $suspends_j$ = ys @ $Ghost_{sb}$ A L R W# zs ∧ a ∈ A
then
obtain A′ L′ R′ W′ ys zs **where**
split-$suspends_j$: $suspends_j$ = ys @ $Ghost_{sb}$ A′ L′ R′ W′# zs
(**is** $suspends_j$ = ?suspends) **and**
a-A′: a ∈ A′
by blast

from sharing-consis [OF j-bound″ ts_{sb}-j]
have sharing-consis-j: sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb_j.
then have A′-R′: A′ ∩ R′ = {}
by (simp add: sharing-consistent-append [of - - ?take-sb_j ?drop-sb_j, simplified]
$suspends_j$ [symmetric] split-$suspends_j$ sharing-consistent-append)
from valid-program-history [OF j-bound″ ts_{sb}-j]
have causal-program-history is_{sbj} sb_j.
then have cph: causal-program-history is_{sbj} ?suspends
apply −
apply (rule causal-program-history-suffix [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp add: split-$suspends_j$)
done

from ts_j neq-i-j j-bound
have ts′-j: ?ts′!j = (hd-prog p_j $suspends_j$, is_j,
j_{sbj} |‘ (dom j_{sbj} − read-tmps $suspends_j$),(),
$\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$, release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
by auto
from valid-last-prog [OF j-bound″ ts_{sb}-j] **have** last-prog: last-prog p_j sb_j = p_j.
then
have lp: last-prog p_j $suspends_j$ = p_j
apply −
apply (rule last-prog-same-append [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

from valid-reads [OF j-bound″ ts_{sb}-j]
have reads-consis-j: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.
from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts_{sb}›
j-bound″ ts_{sb}-j this]
have reads-consis-m-j: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
by (simp add: m $suspends_j$)

from outstanding-non-write-non-vol-reads-drop-disj [OF i-bound j-bound″ neq-i-j ts_{sb}-i ts_{sb}-j]
have outstanding-refs is-$Write_{sb}$?drop-sb ∩ outstanding-refs is-non-volatile-$Read_{sb}$ $suspends_j$ = {}
by (simp add: $suspends_j$)
from reads-consistent-flush-independent [OF this reads-consis-m-j]
have reads-consis-flush-suspend: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) (flush ?drop-sb m) $suspends_j$.

hence reads-consis-ys: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) (flush ?drop-sb m) (ys@[$Ghost_{sb}$ A′ L′ R′ W′])
by (simp add: split-$suspends_j$ reads-consistent-append)

from valid-write-sops [OF j-bound″ ts_{sb}-j]
have ∀sop∈write-sops (?take-sb_j@?suspends). valid-sop sop
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain valid-sops-take: ∀sop∈write-sops ?take-sb_j. valid-sop sop **and**
valid-sops-drop: ∀sop∈write-sops (ys@[$Ghost_{sb}$ A′ L′ R′ W′]). valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound″ ts_{sb}-j]
have distinct-read-tmps (?take-sb_j@$suspends_j$)
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain

read-tmps-take-drop: read-tmps ?take-sb_j ∩ read-tmps $suspends_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps $suspends_j$
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp only: distinct-read-tmps-append)
done

from valid-history [OF j-bound′′ ts_{sb}-j]
have h-consis:
history-consistent j_{sbj} (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof −
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop
h-consis] last-prog-hd-prog
have hist-consis′: history-consistent j_{sbj} (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis-j]
have no-vol-read: outstanding-refs is-volatile-$Read_{sb}$
(ys@[$Ghost_{sb}$ A′ L′ R′ W′]) = {}
by (auto simp add: outstanding-refs-append $suspends_j$ [symmetric]
split-$suspends_j$)

have acq-simp:
acquired True (ys @ [$Ghost_{sb}$ A′ L′ R′ W′])
(acquired True ?take-sb_j $\mathcal{O}_j$) =
acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$) ∪ A′ − R′
by (simp add: acquired-append)

from flush-store-buffer-append [**where** sb=ys@[$Ghost_{sb}$ A′ L′ R′ W′] **and** sb′=zs,
simplified,
OF j-bound-ts′ is_j [simplified split-$suspends_j$] cph [simplified $suspends_j$]
ts′-j [simplified split-$suspends_j$] refl lp [simplified split-$suspends_j$] reads-consis-ys

hist-consis′ [simplified split-suspends$_j$] valid-sops-drop
distinct-read-tmps-drop [simplified split-suspends$_j$]
no-volatile-Read$_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=share ?drop-sb $\mathcal{S}$]
obtain is$_j$′ $\mathcal{R}_j$′ **where**
is$_j$′: instrs zs @ is$_{sbj}$ = is$_j$′ @ prog-instrs zs **and**
steps-ys: (?ts′, flush ?drop-sb m, share ?drop-sb $\mathcal{S}$) $\Rightarrow_d^*$
(?ts′[j:=(last-prog
(hd-prog p$_j$ (Ghost$_{sb}$ A′ L′ R′ W′# zs)) (ys@[Ghost$_{sb}$ A′ L′ R′ W′]),
is$_j$′,
j$_{sbj}$ |' (dom j$_{sbj}$ − read-tmps zs),
(),
$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-Write$_{sb}$ (ys @ [Ghost$_{sb}$ A′ L′ R′ W′]) ≠ {}, acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ∪ A′ − R′,$\mathcal{R}_j$′)],
flush (ys@[Ghost$_{sb}$ A′ L′ R′ W′]) (flush ?drop-sb m),
share (ys@[Ghost$_{sb}$ A′ L′ R′ W′]) (share ?drop-sb $\mathcal{S}$))
(**is** (-,-,-) $\Rightarrow_d^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto simp add: acquired-append)

from i-bound′ **have** i-bound-ys: i < length ?ts-ys
by auto

from i-bound′ neq-i-j
have ts-ys-i: ?ts-ys!i = (p$_{sb}$, is$_{sb}$,j$_{sb}$,(),
$\mathcal{D}_{sb}$, acquired True sb $\mathcal{O}_{sb}$, release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
by simp
note conflict-computation = rtranclp-trans [OF steps-flush-sb steps-ys]

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys).

from safe-delayedE [OF this i-bound-ys ts-ys-i, simplified is$_{sb}$]
have a-unowned:
∀j < length ?ts-ys. i≠j ⟶ (let ($\mathcal{O}_j$) = map owned ?ts-ys!j in a ∉ $\mathcal{O}_j$)
apply cases
apply (auto simp add: Let-def is$_{sb}$)
done
from a-A′ a-unowned [rule-format, of j] neq-i-j j-bound′ A′-R′
show False
by (auto simp add: Let-def)
qed
qed
}
thus ?thesis
by (auto simp add: Let-def)
qed

have A-unused-by-others:
∀j<length (map $\mathcal{O}$-sb ts$_{sb}$). i ≠ j ⟶

(let ($\mathcal{O}_j$, sb_j) = map $\mathcal{O}$-sb ts_{sb}! j
in A ∩ outstanding-refs is-volatile-$Write_{sb}$ sb_j = {})

proof −

{

fix j $\mathcal{O}_j$ sb_j
assume j-bound: j < length (map owned ts_{sb})
assume neq-i-j: i≠j
assume ts_{sb}-j: (map $\mathcal{O}$-sb ts_{sb})!j = ($\mathcal{O}_j$,sb_j)
assume conflict: A ∩ outstanding-refs is-volatile-$Write_{sb}$ sb_j ≠ {}
have False
proof −
from j-bound leq
have j-bound′: j < length (map owned ts)
by auto
from j-bound **have** j-bound″: j < length ts_{sb}
by auto
from j-bound′ **have** j-bound‴: j < length ts
by simp

from conflict **obtain** a′ **where**
a′-in: a′ ∈ A **and**
a′-in-j: a′ ∈ outstanding-refs is-volatile-$Write_{sb}$ sb_j
by auto

let ?take-sb_j = (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)
let ?drop-sb_j = (dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)

from ts-sim [rule-format, OF j-bound″] ts_{sb}-j j-bound″
obtain p_j $suspends_j$ is_{sbj} $\mathcal{D}_{sbj}$ $\mathcal{D}_j$ $\mathcal{R}_j$ j_{sbj} is_j **where**
ts_{sb}-j: ts_{sb} ! j = (p_j,is_{sbj}, j_{sbj}, sb_j,$\mathcal{D}_{sbj}$,$\mathcal{O}_j$,$\mathcal{R}_j$) **and**
$suspends_j$: $suspends_j$ = ?drop-sb_j **and**
is_j: instrs $suspends_j$ @ is_{sbj} = is_j @ prog-instrs $suspends_j$ **and**
$\mathcal{D}_j$: $\mathcal{D}_{sbj}$ = ($\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ sb_j ≠ {}) **and**
ts_j: ts!j = (hd-prog p_j $suspends_j$, is_j,
j_{sbj} |‘ (dom j_{sbj} − read-tmps $suspends_j$),(), $\mathcal{D}_j$,
acquired True ?take-sb_j $\mathcal{O}_j$,
release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
apply (cases ts_{sb}!j)
apply (force simp add: Let-def)
done

have a′ ∈ outstanding-refs is-volatile-$Write_{sb}$ $suspends_j$
proof −
from a′-in-j
have a′ ∈ outstanding-refs is-volatile-$Write_{sb}$ (?take-sb_j @ ?drop-sb_j)
by simp
thus ?thesis
apply (simp only: outstanding-refs-append $suspends_j$)
apply (auto simp add: outstanding-refs-conv dest: set-takeWhileD)
done

qed

from split-volatile-$\text{Write}_{\mathsf{sb}}$-in-outstanding-refs [OF this]
obtain sop v ys zs A' L' R' W' **where**
split-$\text{suspends}_{\mathsf{j}}$: $\text{suspends}_{\mathsf{j}}$ = ys @ $\text{Write}_{\mathsf{sb}}$ True a' sop v A' L' R' W'# zs (**is** $\text{suspends}_{\mathsf{j}}$ = ?suspends)
by blast

from direct-memop-step.WriteVolatile [**where** $j=j_{\mathsf{sb}}$ **and** m=flush ?drop-sb m]
have (Write True a (D, f) A L R W# is_{sb}',
j_{sb}, (), flush ?drop-sb m,$\mathcal{D}_{\mathsf{sb}}$,acquired True sb $\mathcal{O}_{\mathsf{sb}}$,
release sb (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}_{\mathsf{sb}}$,
share ?drop-sb $\mathcal{S}$) $\rightarrow$
(is_{sb}', j_{sb}, (), (flush ?drop-sb m)(a := f j_{sb}), True, acquired True sb $\mathcal{O}_{\mathsf{sb}} \cup$ A $-$ R, Map.empty,
share ?drop-sb $\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L).

from direct-computation.concurrent-step.Memop [OF
i-bound-ts$'$ [simplified is_{sb}] ts$'$-i [simplified is_{sb}] this [simplified is_{sb}]]
have store-step: (?ts$'$, flush ?drop-sb m,share ?drop-sb $\mathcal{S}$) $\Rightarrow_{\mathsf{d}}$
(?ts$'$[i := (p_{sb}, is_{sb}', j_{sb}, (),
True, acquired True sb $\mathcal{O}_{\mathsf{sb}} \cup$ A $-$ R,Map.empty)],
(flush ?drop-sb m)(a := f j_{sb}), share ?drop-sb $\mathcal{S} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
(**is** - $\Rightarrow_{\mathsf{d}}$ (?ts-A, ?m-A, ?share-A))
by (simp add: is_{sb})

from i-bound$'$ **have** i-bound$''$: i < length ?ts-A
by simp

from valid-program-history [OF j-bound$''$ ts_{sb}-j]
have causal-program-history is_{sbj} sb_{j}.
then have cph: causal-program-history is_{sbj} ?suspends
apply $-$
apply (rule causal-program-history-suffix [**where** sb=?take-sb_{j}])
apply (simp only: split-$\text{suspends}_{\mathsf{j}}$ [symmetric] $\text{suspends}_{\mathsf{j}}$)
apply (simp add: split-$\text{suspends}_{\mathsf{j}}$)
done

from ts_{j} neq-i-j j-bound
have ts-A-j: ?ts-A!j = (hd-prog p_{j} (ys @ $\text{Write}_{\mathsf{sb}}$ True a' sop v A' L' R' W'# zs), is_{j},
j_{sbj} |‘ (dom j_{sbj} $-$ read-tmps (ys @ $\text{Write}_{\mathsf{sb}}$ True a' sop v A' L' R' W'# zs)), (), $\mathcal{D}_{\mathsf{j}}$,
acquired True ?take-sb_{j} $\mathcal{O}_{\mathsf{j}}$,release ?take-sb_{j} (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}_{\mathsf{j}}$)
by (simp add: split-$\text{suspends}_{\mathsf{j}}$)

from j-bound$'''$ i-bound$'$ neq-i-j **have** j-bound$''''$: j < length ?ts-A
by simp

from valid-last-prog [OF j-bound$''$ ts_{sb}-j] **have** last-prog: last-prog p_{j} sb_{j} = p_{j}.

then
have lp: last-prog p_j ?suspends = p_j
apply −
apply (rule last-prog-same-append [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

from valid-reads [OF j-bound″ ts_{sb}-j]
have reads-consis: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.

from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts_{sb}› j-bound″
ts_{sb}-j reads-consis]
have reads-consis-m: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
by (simp add: m $suspends_j$)

from outstanding-non-write-non-vol-reads-drop-disj [OF i-bound j-bound″ neq-i-j ts_{sb}-i ts_{sb}-j]
have outstanding-refs is-$Write_{sb}$?drop-sb ∩ outstanding-refs is-non-volatile-$Read_{sb}$ $suspends_j$ = {}
by (simp add: $suspends_j$)
from reads-consistent-flush-independent [OF this reads-consis-m]
have reads-consis-flush-m: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$)
(flush ?drop-sb m) $suspends_j$.

from a-unowned-others [rule-format, OF - neq-i-j] j-bound ts_{sb}-j
obtain a-notin-owns-j: a ∉ acquired True ?take-sb_j $\mathcal{O}_j$ **and** a-unshared: a ∉ all-shared ?take-sb_j
by auto
from a-not-acquired-others [rule-format, OF - neq-i-j] j-bound ts_{sb}-j
have a-not-acquired-j: a ∉ all-acquired sb_j
by auto

from outstanding-non-volatile-refs-owned-or-read-only [OF j-bound″ ts_{sb}-j]
have nvo-j: non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb_j.

have a-no-non-vol-read: a ∉ outstanding-refs is-non-volatile-$Read_{sb}$?drop-sb_j
proof
assume a-in-nvr:a ∈ outstanding-refs is-non-volatile-$Read_{sb}$?drop-sb_j

from reads-consistent-drop [OF reads-consis]
have rc: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) (flush ?take-sb_j m_{sb}) ?drop-sb_j.

from non-volatile-owned-or-read-only-drop [OF nvo-j]
have nvo-j-drop: non-volatile-owned-or-read-only True (share ?take-sb_j $\mathcal{S}_{sb}$)
(acquired True ?take-sb_j $\mathcal{O}_j$)
?drop-sb_j

by simp

from outstanding-refs-non-volatile-Read$_{sb}$-all-acquired [OF rc this a-in-nvr]

have a-owns-acq-ror:
a $\in \mathcal{O}_j \cup$ all-acquired sb$_j$ $\cup$ read-only-reads (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?drop-sb$_j$
by (auto dest!: acquired-all-acquired-in all-acquired-takeWhile-dropWhile-in
simp add: acquired-takeWhile-non-volatile-Write$_{sb}$)

have a-unowned-j: a $\notin \mathcal{O}_j \cup$ all-acquired sb$_j$
proof (cases a $\in \mathcal{O}_j$)
case False **with** a-not-acquired-j **show** ?thesis **by** auto
next
case True
from all-shared-acquired-in [OF True a-unshared] a-notin-owns-j
have False **by** auto **thus** ?thesis **..**
qed
with a-owns-acq-ror
have a-ror: a $\in$ read-only-reads (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?drop-sb$_j$
by auto

with read-only-reads-unowned [OF j-bound$''$ i-bound neq-i-j [symmetric] ts$_{sb}$-j ts$_{sb}$-i]
have a-unowned-sb: a $\notin \mathcal{O}_{sb} \cup$ all-acquired sb
by auto

from sharing-consis [OF j-bound$''$ ts$_{sb}$-j] sharing-consistent-append [of $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb$_j$?drop-sb$_j$]
have consis-j-drop: sharing-consistent (share ?take-sb$_j$ $\mathcal{S}_{sb}$) (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?drop-sb$_j$
by auto

from read-only-reads-read-only [OF nvo-j-drop consis-j-drop] a-ror a-unowned-j
all-acquired-append [of ?take-sb$_j$?drop-sb$_j$] acquired-takeWhile-non-volatile-Write$_{sb}$ [of sb$_j$ $\mathcal{O}_j$]
have a $\in$ read-only (share ?take-sb$_j$ $\mathcal{S}_{sb}$)
by (auto simp add:)
from read-only-share-all-shared [OF this] a-unshared
have a $\in$ read-only $\mathcal{S}_{sb}$
by fastforce

from read-only-unacquired-share [OF read-only-unowned [OF i-bound ts$_{sb}$-i]
weak-sharing-consis [OF i-bound ts$_{sb}$-i] this] a-unowned-sb
have a $\in$ read-only (share sb $\mathcal{S}_{sb}$)
by auto

with a-not-ro **show** False
by simp
qed

with reads-consistent-mem-eq-on-non-volatile-reads [OF - subset-refl reads-consis-flush-m]
have reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) ?m-A $suspends_j$
by (auto simp add: $suspends_j$)

hence reads-consis-m-A-ys: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) ?m-A ys
by (simp add: split-$suspends_j$ reads-consistent-append)

from valid-history [OF j-bound$''$ ts_{sb}-j]
have h-consis:
history-consistent j_{sbj} (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof −
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append$'$ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from valid-write-sops [OF j-bound$''$ ts_{sb}-j]
have ∀sop∈write-sops (?take-sb_j@?suspends). valid-sop sop
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain valid-sops-take: ∀sop∈write-sops ?take-sb_j. valid-sop sop **and**
valid-sops-drop: ∀sop∈write-sops ys. valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound$''$ ts_{sb}-j]
have distinct-read-tmps (?take-sb_j@$suspends_j$)
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb_j ∩ read-tmps $suspends_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps $suspends_j$
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp only: distinct-read-tmps-append)

done

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop h-consis] last-prog-hd-prog
have hist-consis′: history-consistent j_{sbj} (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis]
have no-vol-read: outstanding-refs is-volatile-$Read_{sb}$ ys = {}
by (auto simp add: outstanding-refs-append $suspends_j$ [symmetric] split-$suspends_j$)

from flush-store-buffer-append [
OF j-bound′′′′ is_j [simplified split-$suspends_j$] cph [simplified $suspends_j$]
ts-A-j [simplified split-$suspends_j$] refl lp [simplified split-$suspends_j$] reads-consis-m-A-ys
hist-consis′ [simplified split-$suspends_j$] valid-sops-drop distinct-read-tmps-drop [simplified split-$suspends_j$]
no-volatile-$Read_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=?share-A]
obtain is_j′ $\mathcal{R}_j$′ **where**
is_j′: instrs ($Write_{sb}$ True a′ sop v A′ L′ R′ W′# zs) @ is_{sbj} =
is_j′ @ prog-instrs ($Write_{sb}$ True a′ sop v A′ L′ R′ W′# zs) **and**
steps-ys: (?ts-A, ?m-A, ?share-A) $\Rightarrow_d^*$
(?ts-A[j:= (last-prog (hd-prog p_j ($Write_{sb}$ True a′ sop v A′ L′ R′ W′# zs)) ys,
is_j′,
j_{sbj} |‘ (dom j_{sbj} − read-tmps ($Write_{sb}$ True a′ sop v A′ L′ R′ W′ # zs)),(),
$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ ys ≠ {}, acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$),$\mathcal{R}_j$′)],
flush ys ?m-A,
share ys ?share-A)
(**is** (-,-,-) $\Rightarrow_d^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto)

note conflict-computation = rtranclp-trans [OF rtranclp-r-rtranclp [OF steps-flush-sb, OF store-step] steps-ys]
from cph
have causal-program-history is_{sbj} ((ys @ [$Write_{sb}$ True a′ sop v A′ L′ R′ W′]) @ zs)
by simp
from causal-program-history-suffix [OF this]
have cph′: causal-program-history is_{sbj} zs**.**
interpret $causal_j$: causal-program-history is_{sbj} zs **by** (rule cph′)

from $causal_j$.causal-program-history [of [], simplified, OF refl] is_j′
obtain is_j′′
where is_j′: is_j′ = Write True a′ sop A′ L′ R′ W′#is_j′′ **and**
is_j′′: instrs zs @ is_{sbj} = is_j′′ @ prog-instrs zs
by clarsimp

from j-bound′′′

have j-bound-ys: j < length ?ts-ys
  by auto

from j-bound-ys neq-i-j
have ts-ys-j: ?ts-ys!j=(last-prog (hd-prog p_j (Write$_{sb}$ True a$'$ sop v A$'$ L$'$ R$'$ W$'$# zs)) ys, is$_j$$'$,
    j$_{sbj}$ |‘ (dom j$_{sbj}$ − read-tmps (Write$_{sb}$ True a$'$ sop v A$'$ L$'$ R$'$ W$'$# zs)),(),
  $\mathcal{D}_j$ ∨ outstanding-refs is-volatile-Write$_{sb}$ ys ≠ {},
    acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$),$\mathcal{R}_j$$'$)
  by auto

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys)**.**

from safe-delayedE [OF this j-bound-ys ts-ys-j, simplified is$_j$$'$]
have a-unowned:
∀ i < length ?ts-ys. j≠i ⟶ (let ($\mathcal{O}_i$) = map owned ?ts-ys!i in a$'$ ∉ $\mathcal{O}_i$)
  apply cases
  apply (auto simp add: Let-def is$_{sb}$)
  done
from a$'$-in a-unowned [rule-format, of i] neq-i-j i-bound$'$ A-R
show False
  by (auto simp add: Let-def)
qed
}
thus ?thesis
by (auto simp add: Let-def)
qed

have A-unaquired-by-others:
∀ j<length (map $\mathcal{O}$-sb ts$_{sb}$). i ≠ j ⟶
    (let ($\mathcal{O}_j$, sb$_j$) = map $\mathcal{O}$-sb ts$_{sb}$! j
    in A ∩ all-acquired sb$_j$ = {})
proof −
{
fix j $\mathcal{O}_j$ sb$_j$
assume j-bound: j < length (map owned ts$_{sb}$)
assume neq-i-j: i≠j
assume ts$_{sb}$-j: (map $\mathcal{O}$-sb ts$_{sb}$)!j = ($\mathcal{O}_j$,sb$_j$)
assume conflict: A ∩ all-acquired sb$_j$ ≠ {}
have False
proof −
  from j-bound leq
  have j-bound$'$: j < length (map owned ts)
    by auto
  from j-bound **have** j-bound$''$: j < length ts$_{sb}$
    by auto
  from j-bound$'$ **have** j-bound$'''$: j < length ts
    by simp

from conflict **obtain** a′ **where**
a′-in: a′ ∈ A **and**
a′-in-j: a′ ∈ all-acquired sb_j
by auto

let ?take-sb_j = (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)
let ?drop-sb_j = (dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)

from ts-sim [rule-format, OF j-bound″] ts_{sb}-j j-bound″
obtain p_j $suspends_j$ is_{sbj} $\mathcal{D}_{sbj}$ $\mathcal{D}_j$ $\mathcal{R}_j$ j_{sbj} is_j **where**
ts_{sb}-j: ts_{sb} ! j = (p_j,is_{sbj}, j_{sbj}, sb_j,$\mathcal{D}_{sbj}$,$\mathcal{O}_j$,$\mathcal{R}_j$) **and**
$suspends_j$: $suspends_j$ = ?drop-sb_j **and**
is_j: instrs $suspends_j$ @ is_{sbj} = is_j @ prog-instrs $suspends_j$ **and**
$\mathcal{D}_j$: $\mathcal{D}_{sbj}$ = ($\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ sb_j ≠ {}) **and**
ts_j: ts!j = (hd-prog p_j $suspends_j$, is_j,
j_{sbj} |‘ (dom j_{sbj} − read-tmps $suspends_j$),(),
$\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$,release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
apply (cases ts_{sb}!j)
apply (force simp add: Let-def)
done

from a′-in-j all-acquired-append [of ?take-sb_j ?drop-sb_j]
have a′ ∈ all-acquired ?take-sb_j ∨ a′ ∈ all-acquired $suspends_j$
by (auto simp add: $suspends_j$)
thus False
proof
assume a′ ∈ all-acquired ?take-sb_j
with A-unowned-by-others [rule-format, OF - neq-i-j] ts_{sb}-j j-bound a′-in
show False
by (auto dest: all-acquired-unshared-acquired)
next
assume conflict-drop: a′ ∈ all-acquired $suspends_j$
from split-all-acquired-in [OF conflict-drop]

show False
proof
assume ∃sop a″ v ys zs A L R W.
$suspends_j$ = ys @ $Write_{sb}$ True a″ sop v A L R W# zs ∧ a′ ∈ A
then
obtain a″ sop′ v′ ys zs A′ L′ R′ W′ **where**
split-$suspends_j$: $suspends_j$ = ys @ $Write_{sb}$ True a″ sop′ v′ A′ L′ R′ W′# zs
(**is** $suspends_j$ = ?suspends) **and**
a′-A′: a′ ∈ A′
by auto

from direct-memop-step.WriteVolatile [**where** j=j_{sb} **and** m=flush ?drop-sb m]
have (Write True a (D, f) A L R W # is_{sb}′,
j_{sb}, (), flush ?drop-sb m ,$\mathcal{D}_{sb}$, acquired True sb $\mathcal{O}_{sb}$,
release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$,
share ?drop-sb $\mathcal{S}$) →

$(is_{sb}{}', j_{sb}, (), (\text{flush ?drop-sb m})(a := f\ j_{sb}), \text{True}, \text{acquired True sb } \mathcal{O}_{sb} \cup A - R, \text{Map.empty},$
$\text{share ?drop-sb } \mathcal{S} \oplus_W R \ominus_A L)$.

from direct-computation.concurrent-step.Memop [OF
i-bound-ts′ [simplified is_{sb}] ts′-i [simplified is_{sb}] this [simplified is_{sb}]]

have store-step: (?ts′, flush ?drop-sb m, share ?drop-sb $\mathcal{S}$) $\Rightarrow_d$
(?ts′[i := (p_{sb}, is_{sb}′,
j_{sb}, (),True, acquired True sb $\mathcal{O}_{sb} \cup A - R$,Map.empty)],
(flush ?drop-sb m)(a := f j_{sb}),share ?drop-sb $\mathcal{S} \oplus_W R \ominus_A L$)
(**is** - $\Rightarrow_d$ (?ts-A, ?m-A, ?share-A))
by (simp add: is_{sb})

from i-bound′ **have** i-bound″: i < length ?ts-A
by simp

from valid-program-history [OF j-bound″ ts_{sb}-j]
have causal-program-history is_{sbj} sb_j.
then have cph: causal-program-history is_{sbj} ?suspends
apply −
apply (rule causal-program-history-suffix [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp add: split-$suspends_j$)
done

from ts_j neq-i-j j-bound
have ts-A-j: ?ts-A!j = (hd-prog p_j (ys @ $Write_{sb}$ True a″ sop′ v′ A′ L′ R′ W′# zs), is_j,
j_{sbj} |‘ (dom j_{sbj} − read-tmps (ys @ $Write_{sb}$ True a″ sop′ v′ A′ L′ R′ W′# zs)), (), $\mathcal{D}_j$,
acquired True ?take-sb_j $\mathcal{O}_j$,release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
by (simp add: split-$suspends_j$)

from j-bound‴ i-bound′ neq-i-j **have** j-bound⁗: j < length ?ts-A
by simp

from valid-last-prog [OF j-bound″ ts_{sb}-j] **have** last-prog: last-prog p_j sb_j = p_j.
then
have lp: last-prog p_j ?suspends = p_j
apply −
apply (rule last-prog-same-append [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

from valid-reads [OF j-bound″ ts_{sb}-j]
have reads-consis: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.

from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts_{sb}›
j-bound″
ts_{sb}-j reads-consis]
have reads-consis-m: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
by (simp add: m $suspends_j$)

from outstanding-non-write-non-vol-reads-drop-disj [OF i-bound j-bound″ neq-i-j ts_{sb}-i ts_{sb}-j]
have outstanding-refs is-$Write_{sb}$?drop-sb ∩ outstanding-refs is-non-volatile-$Read_{sb}$ $suspends_j$ = {}
by (simp add: $suspends_j$)
from reads-consistent-flush-independent [OF this reads-consis-m]
have reads-consis-flush-m: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$)
(flush ?drop-sb m) $suspends_j$.

from a-unowned-others [rule-format, OF - neq-i-j] j-bound ts_{sb}-j
obtain a-notin-owns-j: a ∉ acquired True ?take-sb_j $\mathcal{O}_j$ **and** a-unshared: a ∉ all-shared ?take-sb_j
by auto
from a-not-acquired-others [rule-format, OF - neq-i-j] j-bound ts_{sb}-j
have a-not-acquired-j: a ∉ all-acquired sb_j
by auto

from outstanding-non-volatile-refs-owned-or-read-only [OF j-bound″ ts_{sb}-j]
have nvo-j: non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb_j.

have a-no-non-vol-read: a ∉ outstanding-refs is-non-volatile-$Read_{sb}$?drop-sb_j
proof
assume a-in-nvr:a ∈ outstanding-refs is-non-volatile-$Read_{sb}$?drop-sb_j

from reads-consistent-drop [OF reads-consis]
have rc: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) (flush ?take-sb_j m_{sb}) ?drop-sb_j.

from non-volatile-owned-or-read-only-drop [OF nvo-j]
have nvo-j-drop: non-volatile-owned-or-read-only True (share ?take-sb_j $\mathcal{S}_{sb}$)
(acquired True ?take-sb_j $\mathcal{O}_j$)
?drop-sb_j
by simp

from outstanding-refs-non-volatile-$Read_{sb}$-all-acquired [OF rc this a-in-nvr]

have a-owns-acq-ror:
a ∈ $\mathcal{O}_j$ ∪ all-acquired sb_j ∪ read-only-reads (acquired True ?take-sb_j $\mathcal{O}_j$) ?drop-sb_j
by (auto dest!: acquired-all-acquired-in all-acquired-takeWhile-dropWhile-in
simp add: acquired-takeWhile-non-volatile-$Write_{sb}$)
have a-unowned-j: a ∉ $\mathcal{O}_j$ ∪ all-acquired sb_j
proof (cases a ∈ $\mathcal{O}_j$)
case False **with** a-not-acquired-j **show** ?thesis **by** auto

next
case True
from all-shared-acquired-in [OF True a-unshared] a-notin-owns-j
have False **by** auto **thus** ?thesis **..**
qed

with a-owns-acq-ror
have a-ror: a $\in$ read-only-reads (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?drop-sb$_j$
by auto

with read-only-reads-unowned [OF j-bound$''$ i-bound neq-i-j [symmetric] ts$_{sb}$-j ts$_{sb}$-i]
have a-unowned-sb: a $\notin$ $\mathcal{O}_{sb}$ $\cup$ all-acquired sb
by auto

from sharing-consis [OF j-bound$''$ ts$_{sb}$-j] sharing-consistent-append [of $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb$_j$?drop-sb$_j$]
have consis-j-drop: sharing-consistent (share ?take-sb$_j$ $\mathcal{S}_{sb}$) (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?drop-sb$_j$
by auto

from read-only-reads-read-only [OF nvo-j-drop consis-j-drop] a-ror a-unowned-j
all-acquired-append [of ?take-sb$_j$?drop-sb$_j$] acquired-takeWhile-non-volatile-Write$_{sb}$ [of sb$_j$ $\mathcal{O}_j$]
have a $\in$ read-only (share ?take-sb$_j$ $\mathcal{S}_{sb}$)
by (auto)
from read-only-share-all-shared [OF this] a-unshared
have a $\in$ read-only $\mathcal{S}_{sb}$
by fastforce

from read-only-unacquired-share [OF read-only-unowned [OF i-bound ts$_{sb}$-i]
weak-sharing-consis [OF i-bound ts$_{sb}$-i] this] a-unowned-sb
have a $\in$ read-only (share sb $\mathcal{S}_{sb}$)
by auto

with a-not-ro **show** False
by simp
qed
with reads-consistent-mem-eq-on-non-volatile-reads [OF - subset-refl reads-consis-flush-m]
have reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?m-A suspends$_j$
by (auto simp add: suspends$_j$)

hence reads-consis-m-A-ys: reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?m-A ys
by (simp add: split-suspends$_j$ reads-consistent-append)

from valid-history [OF j-bound$''$ ts$_{sb}$-j]
have h-consis:

history-consistent $j_{\mathsf{sb}j}$ (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof −
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from valid-write-sops [OF j-bound″ ts_{sb}-j]
have ∀sop∈write-sops (?take-sb_j@?suspends). valid-sop sop
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain valid-sops-take: ∀sop∈write-sops ?take-sb_j. valid-sop sop **and**
valid-sops-drop: ∀sop∈write-sops ys. valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound″ ts_{sb}-j]
have distinct-read-tmps (?take-sb_j@$suspends_j$)
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb_j ∩ read-tmps $suspends_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps $suspends_j$
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp only: distinct-read-tmps-append)
done

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop h-consis]

last-prog-hd-prog
have hist-consis′: history-consistent $j_{\mathsf{sb}j}$ (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis]
have no-vol-read: outstanding-refs is-volatile-$Read_{\mathsf{sb}}$ ys = {}
by (auto simp add: outstanding-refs-append $suspends_j$ [symmetric]
split-$suspends_j$)

from flush-store-buffer-append [
OF j-bound'''' is_j [simplified split-suspends$_j$] cph [simplified suspends$_j$]
ts-A-j [simplified split-suspends$_j$] refl lp [simplified split-suspends$_j$] reads-consis-m-A-ys
hist-consis' [simplified split-suspends$_j$] valid-sops-drop distinct-read-tmps-drop [simplified split-suspends$_j$]
no-volatile-Read$_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=?share-A]
obtain is_j' $\mathcal{R}_j'$ **where**
is_j': instrs (Write$_{sb}$ True a'' sop' v' A' L' R' W' # zs) @ is_{sbj} =
is_j' @ prog-instrs (Write$_{sb}$ True a'' sop' v' A' L' R' W' # zs) **and**
steps-ys: (?ts-A, ?m-A, ?share-A) $\Rightarrow_d^*$
(?ts-A[j:= (last-prog (hd-prog p_j (Write$_{sb}$ True a'' sop' v' A' L' R' W' # zs)) ys,
is_j',
j_{sbj} |' (dom j_{sbj} − read-tmps (Write$_{sb}$ True a'' sop' v' A' L' R' W' # zs)),(),
$\mathcal{D}_j$ $\vee$ outstanding-refs is-volatile-Write$_{sb}$ ys $\neq$ {}, acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$),$\mathcal{R}_j'$)],
flush ys ?m-A, share ys ?share-A)
(**is** (-,-,-) $\Rightarrow_d^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto)

note conflict-computation = rtranclp-trans [OF rtranclp-r-rtranclp [OF steps-flush-sb, OF store-step] steps-ys]
from cph
have causal-program-history is_{sbj} ((ys @ [Write$_{sb}$ True a'' sop' v' A' L' R' W']) @ zs)
by simp
from causal-program-history-suffix [OF this]
have cph': causal-program-history is_{sbj} zs.
interpret causal$_j$: causal-program-history is_{sbj} zs **by** (rule cph')

from causal$_j$.causal-program-history [of [], simplified, OF refl] is_j'
obtain is_j''
where is_j': is_j' = Write True a'' sop' A' L' R' W'#is_j'' **and**
is_j'': instrs zs @ is_{sbj} = is_j'' @ prog-instrs zs
by clarsimp

from j-bound'''
have j-bound-ys: j < length ?ts-ys
by auto

from j-bound-ys neq-i-j
have ts-ys-j: ?ts-ys!j=(last-prog (hd-prog p_j (Write$_{sb}$ True a'' sop' v' A' L' R' W'# zs)) ys, is_j',
j_{sbj} |' (dom j_{sbj} − read-tmps (Write$_{sb}$ True a'' sop' v' A' L' R' W'# zs)),(),$\mathcal{D}_j$ $\vee$ outstanding-refs is-volatile-Write$_{sb}$ ys $\neq$ {},
acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$),$\mathcal{R}_j'$)
by auto

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys).

from safe-delayedE [OF this j-bound-ys ts-ys-j, simplified $is_j{}'$]
have A′-unowned:
$\forall$ i < length ?ts-ys. j$\neq$i $\longrightarrow$ (let ($\mathcal{O}_i$) = map owned ?ts-ys!i in A′ $\cap$ $\mathcal{O}_i$ = {})
apply cases
apply (fastforce simp add: Let-def is_{sb})+
done
from a′-in a′-A′ A′-unowned [rule-format, of i] neq-i-j i-bound′ A-R
show False
by (auto simp add: Let-def)
next
assume $\exists$ A L R W ys zs.
$suspends_j$ = ys @ $Ghost_{sb}$ A L R W # zs $\wedge$ a′ $\in$ A
then
obtain ys zs A′ L′ R′ W′ **where**
split-$suspends_j$: $suspends_j$ = ys @ $Ghost_{sb}$ A′ L′ R′ W′# zs (**is** $suspends_j$ = ?suspends)
and
a′-A′: a′ $\in$ A′
by auto

from direct-memop-step.WriteVolatile [**where** j=j_{sb} **and** m=flush ?drop-sb m]
have (Write True a (D, f) A L R W# $is_{sb}{}'$,
j_{sb}, (), flush ?drop-sb m,$\mathcal{D}_{sb}$,acquired True sb $\mathcal{O}_{sb}$,
release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$,
share ?drop-sb $\mathcal{S}$) $\rightarrow$
($is_{sb}{}'$, j_{sb}, (), (flush ?drop-sb m)(a := f j_{sb}), True, acquired True sb $\mathcal{O}_{sb}$ $\cup$ A $-$ R, Map.empty,
share ?drop-sb $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L).

from direct-computation.concurrent-step.Memop [OF
i-bound-ts′ [simplified is_{sb}] ts′-i [simplified is_{sb}] this [simplified is_{sb}]]
have store-step: (?ts′, flush ?drop-sb m, share ?drop-sb $\mathcal{S}$) $\Rightarrow_d$
(?ts′[i := (p_{sb}, $is_{sb}{}'$,
j_{sb}, (), True, acquired True sb $\mathcal{O}_{sb}$ $\cup$ A $-$ R,Map.empty)],
(flush ?drop-sb m)(a := f j_{sb}),share ?drop-sb $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)
(**is** - $\Rightarrow_d$ (?ts-A, ?m-A, ?share-A))
by (simp add: is_{sb})

from i-bound′ **have** i-bound″: i < length ?ts-A
by simp

from valid-program-history [OF j-bound″ ts_{sb}-j]
have causal-program-history is_{sbj} sb_j.
then have cph: causal-program-history is_{sbj} ?suspends
apply $-$
apply (rule causal-program-history-suffix [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)

apply (simp add: split-suspends$_j$)
done

from ts$_j$ neq-i-j j-bound
have ts-A-j: ?ts-A!j = (hd-prog p$_j$ (ys @ Ghost$_{sb}$ A$'$ L$'$ R$'$ W$'$# zs), is$_j$,
j$_{sbj}$ |' (dom j$_{sbj}$ − read-tmps (ys @ Ghost$_{sb}$ A$'$ L$'$ R$'$ W$'$# zs)), (),$\mathcal{D}_j$,
acquired True ?take-sb$_j$ $\mathcal{O}_j$,release ?take-sb$_j$ (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
by (simp add: split-suspends$_j$)

from j-bound$'''$ i-bound$'$ neq-i-j **have** j-bound$''''$: j < length ?ts-A
by simp

from valid-last-prog [OF j-bound$''$ ts$_{sb}$-j] **have** last-prog: last-prog p$_j$ sb$_j$ = p$_j$.
then
have lp: last-prog p$_j$?suspends = p$_j$
apply −
apply (rule last-prog-same-append [**where** sb=?take-sb$_j$])
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
apply simp
done

from valid-reads [OF j-bound$''$ ts$_{sb}$-j]
have reads-consis: reads-consistent False $\mathcal{O}_j$ m$_{sb}$ sb$_j$.

from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts$_{sb}$›
j-bound$''$
ts$_{sb}$-j reads-consis]
have reads-consis-m: reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$) m suspends$_j$
by (simp add: m suspends$_j$)

from outstanding-non-write-non-vol-reads-drop-disj [OF i-bound j-bound$''$ neq-i-j ts$_{sb}$-i ts$_{sb}$-j]
have outstanding-refs is-Write$_{sb}$?drop-sb ∩ outstanding-refs is-non-volatile-Read$_{sb}$ suspends$_j$ = {}
by (simp add: suspends$_j$)
from reads-consistent-flush-independent [OF this reads-consis-m]
have reads-consis-flush-m: reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$)
(flush ?drop-sb m) suspends$_j$.

from a-unowned-others [rule-format, OF - neq-i-j] j-bound ts$_{sb}$-j
obtain a-notin-owns-j: a ∉ acquired True ?take-sb$_j$ $\mathcal{O}_j$ **and** a-unshared: a ∉ all-shared ?take-sb$_j$
by auto
from a-not-acquired-others [rule-format, OF - neq-i-j] j-bound ts$_{sb}$-j
have a-not-acquired-j: a ∉ all-acquired sb$_j$
by auto

from outstanding-non-volatile-refs-owned-or-read-only [OF j-bound$''$ ts$_{sb}$-j]

have nvo-j: non-volatile-owned-or-read-only False $\mathcal{S}_{\mathsf{sb}}$ $\mathcal{O}_{\mathsf{j}}$ sb_{j}.

have a-no-non-vol-read: a $\notin$ outstanding-refs is-non-volatile-Read$_{\mathsf{sb}}$?drop-sb$_{\mathsf{j}}$
proof
assume a-in-nvr:a $\in$ outstanding-refs is-non-volatile-Read$_{\mathsf{sb}}$?drop-sb$_{\mathsf{j}}$

from reads-consistent-drop [OF reads-consis]
have rc: reads-consistent True (acquired True ?take-sb$_{\mathsf{j}}$ $\mathcal{O}_{\mathsf{j}}$) (flush ?take-sb$_{\mathsf{j}}$ m$_{\mathsf{sb}}$) ?drop-sb$_{\mathsf{j}}$.

from non-volatile-owned-or-read-only-drop [OF nvo-j]
have nvo-j-drop: non-volatile-owned-or-read-only True (share ?take-sb$_{\mathsf{j}}$ $\mathcal{S}_{\mathsf{sb}}$)
(acquired True ?take-sb$_{\mathsf{j}}$ $\mathcal{O}_{\mathsf{j}}$)
?drop-sb$_{\mathsf{j}}$
by simp

from outstanding-refs-non-volatile-Read$_{\mathsf{sb}}$-all-acquired [OF rc this a-in-nvr]

have a-owns-acq-ror:
a $\in$ $\mathcal{O}_{\mathsf{j}}$ $\cup$ all-acquired sb$_{\mathsf{j}}$ $\cup$ read-only-reads (acquired True ?take-sb$_{\mathsf{j}}$ $\mathcal{O}_{\mathsf{j}}$) ?drop-sb$_{\mathsf{j}}$
by (auto dest!: acquired-all-acquired-in all-acquired-takeWhile-dropWhile-in
simp add: acquired-takeWhile-non-volatile-Write$_{\mathsf{sb}}$)

have a-unowned-j: a $\notin$ $\mathcal{O}_{\mathsf{j}}$ $\cup$ all-acquired sb$_{\mathsf{j}}$
proof (cases a $\in$ $\mathcal{O}_{\mathsf{j}}$)
case False **with** a-not-acquired-j **show** ?thesis **by** auto
next
case True
from all-shared-acquired-in [OF True a-unshared] a-notin-owns-j
have False **by** auto **thus** ?thesis **..**
qed

with a-owns-acq-ror
have a-ror: a $\in$ read-only-reads (acquired True ?take-sb$_{\mathsf{j}}$ $\mathcal{O}_{\mathsf{j}}$) ?drop-sb$_{\mathsf{j}}$
by auto

with read-only-reads-unowned [OF j-bound$''$ i-bound neq-i-j [symmetric] ts$_{\mathsf{sb}}$-j ts$_{\mathsf{sb}}$-i]
have a-unowned-sb: a $\notin$ $\mathcal{O}_{\mathsf{sb}}$ $\cup$ all-acquired sb
by auto

from sharing-consis [OF j-bound$''$ ts$_{\mathsf{sb}}$-j] sharing-consistent-append [of $\mathcal{S}_{\mathsf{sb}}$ $\mathcal{O}_{\mathsf{j}}$?take-sb$_{\mathsf{j}}$?drop-sb$_{\mathsf{j}}$]
have consis-j-drop: sharing-consistent (share ?take-sb$_{\mathsf{j}}$ $\mathcal{S}_{\mathsf{sb}}$) (acquired True ?take-sb$_{\mathsf{j}}$ $\mathcal{O}_{\mathsf{j}}$) ?drop-sb$_{\mathsf{j}}$
by auto

from read-only-reads-read-only [OF nvo-j-drop consis-j-drop] a-ror a-unowned-j
all-acquired-append [of ?take-sb$_{\mathsf{j}}$?drop-sb$_{\mathsf{j}}$] acquired-takeWhile-non-volatile-Write$_{\mathsf{sb}}$ [of sb$_{\mathsf{j}}$ $\mathcal{O}_{\mathsf{j}}$]
have a $\in$ read-only (share ?take-sb$_{\mathsf{j}}$ $\mathcal{S}_{\mathsf{sb}}$)

by (auto)
from read-only-share-all-shared [OF this] a-unshared
have a ∈ read-only $\mathcal{S}_{\mathsf{sb}}$
by fastforce

from read-only-unacquired-share [OF read-only-unowned [OF i-bound ts_{sb}-i]
weak-sharing-consis [OF i-bound ts_{sb}-i] this] a-unowned-sb
have a ∈ read-only (share sb $\mathcal{S}_{\mathsf{sb}}$)
by auto

with a-not-ro **show** False
by simp
qed

with reads-consistent-mem-eq-on-non-volatile-reads [OF - subset-refl reads-consis-flush-m]
have reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) ?m-A $suspends_j$
by (auto simp add: $suspends_j$)

hence reads-consis-m-A-ys: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) ?m-A ys
by (simp add: split-$suspends_j$ reads-consistent-append)

from valid-history [OF j-bound″ ts_{sb}-j]
have h-consis:
history-consistent $j_{\mathsf{sb}j}$ (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof −
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from valid-write-sops [OF j-bound″ ts_{sb}-j]
have ∀sop∈write-sops (?take-sb_j@?suspends). valid-sop sop

by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
then obtain valid-sops-take: $\forall$sop$\in$write-sops ?take-sb$_j$. valid-sop sop **and**
valid-sops-drop: $\forall$sop$\in$write-sops ys. valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound$''$ ts$_{\mathsf{sb}}$-j]
have distinct-read-tmps (?take-sb$_j$@suspends$_j$)
by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb$_j$ $\cap$ read-tmps suspends$_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps suspends$_j$
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
apply (simp only: distinct-read-tmps-append)
done

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop h-consis]

last-prog-hd-prog
have hist-consis$'$: history-consistent j$_{\mathsf{sbj}}$ (hd-prog p$_j$ suspends$_j$) suspends$_j$
by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis]
have no-vol-read: outstanding-refs is-volatile-Read$_{\mathsf{sb}}$ ys = {}
by (auto simp add: outstanding-refs-append suspends$_j$ [symmetric]
split-suspends$_j$)

from flush-store-buffer-append [
OF j-bound$''''$ is$_j$ [simplified split-suspends$_j$] cph [simplified suspends$_j$]
ts-A-j [simplified split-suspends$_j$] refl lp [simplified split-suspends$_j$] reads-consis-m-A-ys
hist-consis$'$ [simplified split-suspends$_j$] valid-sops-drop distinct-read-tmps-drop [simplified split-suspends$_j$]
no-volatile-Read$_{\mathsf{sb}}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=?share-A]
obtain is$_j{}'$ $\mathcal{R}_j{}'$ **where**
is$_j{}'$: instrs (Ghost$_{\mathsf{sb}}$ A$'$ L$'$ R$'$ W$'$ # zs) @ is$_{\mathsf{sbj}}$ =
is$_j{}'$ @ prog-instrs (Ghost$_{\mathsf{sb}}$ A$'$ L$'$ R$'$ W$'$# zs) **and**
steps-ys: (?ts-A, ?m-A, ?share-A) $\Rightarrow_{\mathsf{d}}{}^*$
(?ts-A[j:= (last-prog (hd-prog p$_j$ (Ghost$_{\mathsf{sb}}$ A$'$ L$'$ R$'$ W$'$# zs)) ys,
is$_j{}'$,
j$_{\mathsf{sbj}}$ |‘ (dom j$_{\mathsf{sbj}}$ − read-tmps (Ghost$_{\mathsf{sb}}$ A$'$ L$'$ R$'$ W$'$# zs)),(),
$\mathcal{D}_j$ $\vee$ outstanding-refs is-volatile-Write$_{\mathsf{sb}}$ ys $\neq$ {}, acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$),$\mathcal{R}_j{}'$)],
flush ys ?m-A,
share ys ?share-A)
(**is** (-,-,-) $\Rightarrow_{\mathsf{d}}{}^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto)

note conflict-computation = rtranclp-trans [OF rtranclp-r-rtranclp [OF steps-flush-sb, OF store-step] steps-ys]
from cph
have causal-program-history is_{sbj} ((ys @ [$Ghost_{sb}$ A′ L′ R′ W′]) @ zs)
by simp
from causal-program-history-suffix [OF this]
have cph′: causal-program-history is_{sbj} zs.
interpret $causal_j$: causal-program-history is_{sbj} zs **by** (rule cph′)

from $causal_j$.causal-program-history [of [], simplified, OF refl] is_j′
obtain is_j″
where is_j′: is_j′ = Ghost A′ L′ R′ W′#is_j″ **and**
is_j″: instrs zs @ is_{sbj} = is_j″ @ prog-instrs zs
by clarsimp

from j-bound‴
have j-bound-ys: j < length ?ts-ys
by auto

from j-bound-ys neq-i-j
have ts-ys-j: ?ts-ys!j=(last-prog (hd-prog p_j ($Ghost_{sb}$ A′ L′ R′ W′# zs)) ys, is_j′,
j_{sbj} |‘ (dom j_{sbj} − read-tmps ($Write_{sb}$ True a″ sop′ v′ A′ L′ R′ W′# zs)),(),$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ ys ≠ {},
acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$),$\mathcal{R}_j$′)
by auto

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys).

from safe-delayedE [OF this j-bound-ys ts-ys-j, simplified is_j′]
have A′-unowned:
∀i < length ?ts-ys. j≠i ⟶ (let ($\mathcal{O}_i$) = map owned ?ts-ys!i in A′ ∩ $\mathcal{O}_i$ = {})
apply cases
apply (fastforce simp add: Let-def is_{sb})+
done
from a′-in a′-A′ A′-unowned [rule-format, of i] neq-i-j i-bound′ A-R
show False
by (auto simp add: Let-def)
qed
qed
qed
}
thus ?thesis
by (auto simp add: Let-def)
qed

have A-no-read-only-reads-by-others:
∀j<length (map $\mathcal{O}$-sb ts_{sb}). i ≠ j ⟶
(let ($\mathcal{O}_j$, sb_j) = map $\mathcal{O}$-sb ts_{sb}! j

in A ∩ read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$)
$\mathcal{O}_j$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$) = {})
proof −
{
fix j $\mathcal{O}_j$ sb$_j$
assume j-bound: j < length (map $\mathcal{O}$-sb ts$_{sb}$)
assume neq-i-j: i≠j
assume ts$_{sb}$-j: (map $\mathcal{O}$-sb ts$_{sb}$)!j = ($\mathcal{O}_j$,sb$_j$)
let ?take-sb$_j$ = (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$)
let ?drop-sb$_j$ = (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$)

assume conflict: A ∩ read-only-reads (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?drop-sb$_j$ ≠ {}
have False
proof −
from j-bound leq
have j-bound′: j < length (map owned ts)
by auto
from j-bound **have** j-bound′′: j < length ts$_{sb}$
by auto
from j-bound′ **have** j-bound′′′: j < length ts
by simp

from conflict **obtain** a′ **where**
a′-in: a′ ∈ A **and**
a′-in-j: a′ ∈ read-only-reads (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?drop-sb$_j$
by auto

from ts-sim [rule-format, OF j-bound′′] ts$_{sb}$-j j-bound′′
obtain p$_j$ suspends$_j$ is$_{sbj}$ $\mathcal{D}_{sbj}$ $\mathcal{D}_j$ $\mathcal{R}_j$ j$_{sbj}$ is$_j$ **where**
ts$_{sb}$-j: ts$_{sb}$! j = (p$_j$,is$_{sbj}$, j$_{sbj}$, sb$_j$,$\mathcal{D}_{sbj}$,$\mathcal{O}_j$,$\mathcal{R}_j$) **and**
suspends$_j$: suspends$_j$ = ?drop-sb$_j$ **and**
is$_j$: instrs suspends$_j$ @ is$_{sbj}$ = is$_j$ @ prog-instrs suspends$_j$ **and**
$\mathcal{D}_j$: $\mathcal{D}_{sbj}$ = ($\mathcal{D}_j$ ∨ outstanding-refs is-volatile-Write$_{sb}$ sb$_j$ ≠ {}) **and**
ts$_j$: ts!j = (hd-prog p$_j$ suspends$_j$, is$_j$,
j$_{sbj}$ |‘ (dom j$_{sbj}$ − read-tmps suspends$_j$),(), $\mathcal{D}_j$, acquired True ?take-sb$_j$ $\mathcal{O}_j$,release
?take-sb$_j$ (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
apply (cases ts$_{sb}$!j)
apply (force simp add: Let-def)
done

from split-in-read-only-reads [OF a′-in-j [simplified suspends$_j$ [symmetric]]]
obtain t v ys zs **where**
split-suspends$_j$: suspends$_j$ = ys @ Read$_{sb}$ False a′ t v# zs (**is** suspends$_j$ = ?suspends)
and
a′-unacq: a′ ∉ acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$)
by blast

from direct-memop-step.WriteVolatile [**where** j=j_{sb} **and** m=flush ?drop-sb m]
have (Write True a (D, f) A L R W# is_{sb}',
j_{sb}, (), flush ?drop-sb m, $\mathcal{D}_{sb}$,acquired True sb $\mathcal{O}_{sb}$,
release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$, share ?drop-sb $\mathcal{S}$) $\rightarrow$
(is_{sb}', j_{sb}, (), (flush ?drop-sb m)(a := f j_{sb}), True, acquired True sb $\mathcal{O}_{sb}$ $\cup$
A $-$ R,Map.empty,
share ?drop-sb $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L).

from direct-computation.concurrent-step.Memop [OF
i-bound-ts′ [simplified is_{sb}] ts′-i [simplified is_{sb}] this [simplified is_{sb}]]
have store-step: (?ts′, flush ?drop-sb m, share ?drop-sb $\mathcal{S}$) $\Rightarrow_d$
(?ts′[i := (p_{sb}, is_{sb}', j_{sb}, (),
True, acquired True sb $\mathcal{O}_{sb}$ $\cup$ A $-$ R,Map.empty)],
(flush ?drop-sb m)(a := f j_{sb}),share ?drop-sb $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)
(**is** - $\Rightarrow_d$ (?ts-A, ?m-A, ?share-A))
by (simp add: is_{sb})

from i-bound′ **have** i-bound″: i < length ?ts-A
by simp

from valid-program-history [OF j-bound″ ts_{sb}-j]
have causal-program-history is_{sbj} sb_j.
then have cph: causal-program-history is_{sbj} ?suspends
apply $-$
apply (rule causal-program-history-suffix [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp add: split-$suspends_j$)
done

from ts_j neq-i-j j-bound
have ts-A-j: ?ts-A!j = (hd-prog p_j (ys @ $Read_{sb}$ False a′ t v# zs), is_j,
j_{sbj} |‘ (dom j_{sbj} $-$ read-tmps (ys @ $Read_{sb}$ False a′ t v# zs)), (), $\mathcal{D}_j$,
acquired True ?take-sb_j $\mathcal{O}_j$,release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
by (simp add: split-$suspends_j$)

from j-bound‴ i-bound′ neq-i-j **have** j-bound⁗: j < length ?ts-A
by simp

from valid-last-prog [OF j-bound″ ts_{sb}-j] **have** last-prog: last-prog p_j sb_j = p_j.
then
have lp: last-prog p_j ?suspends = p_j
apply $-$
apply (rule last-prog-same-append [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

from valid-reads [OF j-bound″ ts_{sb}-j]

have reads-consis: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.

from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts_{sb}› j-bound''
ts_{sb}-j reads-consis]
have reads-consis-m: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
by (simp add: m $suspends_j$)

from outstanding-non-write-non-vol-reads-drop-disj [OF i-bound j-bound'' neq-i-j ts_{sb}-i ts_{sb}-j]
have outstanding-refs is-$Write_{sb}$?drop-sb ∩ outstanding-refs is-non-volatile-$Read_{sb}$ $suspends_j$ = {}
by (simp add: $suspends_j$)
from reads-consistent-flush-independent [OF this reads-consis-m]
have reads-consis-flush-m: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$)
(flush ?drop-sb m) $suspends_j$.

from a-unowned-others [rule-format, OF j-bound'' neq-i-j] j-bound ts_{sb}-j
obtain a-notin-owns-j: a ∉ acquired True ?take-sb_j $\mathcal{O}_j$ **and** a-unshared: a ∉ all-shared ?take-sb_j
by auto
from a-not-acquired-others [rule-format, OF j-bound neq-i-j] j-bound ts_{sb}-j
have a-not-acquired-j: a ∉ all-acquired sb_j
by auto

from outstanding-non-volatile-refs-owned-or-read-only [OF j-bound'' ts_{sb}-j]
have nvo-j: non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb_j.

have a-no-non-vol-read: a ∉ outstanding-refs is-non-volatile-$Read_{sb}$?drop-sb_j
proof
assume a-in-nvr:a ∈ outstanding-refs is-non-volatile-$Read_{sb}$?drop-sb_j

from reads-consistent-drop [OF reads-consis]
have rc: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) (flush ?take-sb_j m_{sb}) ?drop-sb_j.

from non-volatile-owned-or-read-only-drop [OF nvo-j]
have nvo-j-drop: non-volatile-owned-or-read-only True (share ?take-sb_j $\mathcal{S}_{sb}$)
(acquired True ?take-sb_j $\mathcal{O}_j$)
?drop-sb_j
by simp

from outstanding-refs-non-volatile-$Read_{sb}$-all-acquired [OF rc this a-in-nvr]

have a-owns-acq-ror:
a ∈ $\mathcal{O}_j$ ∪ all-acquired sb_j ∪ read-only-reads (acquired True ?take-sb_j $\mathcal{O}_j$) ?drop-sb_j
by (auto dest!: acquired-all-acquired-in all-acquired-takeWhile-dropWhile-in simp add: acquired-takeWhile-non-volatile-$Write_{sb}$)

have a-unowned-j: a ∉ $\mathcal{O}_j$ ∪ all-acquired sb_j

proof (cases a ∈ $\mathcal{O}_j$)
  case False **with** a-not-acquired-j **show** ?thesis **by** auto
next
  case True
  from all-shared-acquired-in [OF True a-unshared] a-notin-owns-j
  have False **by** auto **thus** ?thesis **..**
qed

with a-owns-acq-ror
have a-ror: a ∈ read-only-reads (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?drop-sb$_j$
  by auto

with read-only-reads-unowned [OF j-bound'' i-bound neq-i-j [symmetric] ts$_{sb}$-j ts$_{sb}$-i]
have a-unowned-sb: a ∉ $\mathcal{O}_{sb}$ ∪ all-acquired sb
  by auto

from sharing-consis [OF j-bound'' ts$_{sb}$-j] sharing-consistent-append [of $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb$_j$?drop-sb$_j$]
have consis-j-drop: sharing-consistent (share ?take-sb$_j$ $\mathcal{S}_{sb}$) (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?drop-sb$_j$
  by auto

from read-only-reads-read-only [OF nvo-j-drop consis-j-drop] a-ror a-unowned-j
  all-acquired-append [of ?take-sb$_j$?drop-sb$_j$] acquired-takeWhile-non-volatile-Write$_{sb}$ [of sb$_j$ $\mathcal{O}_j$]
have a ∈ read-only (share ?take-sb$_j$ $\mathcal{S}_{sb}$)
  by (auto)
from read-only-share-all-shared [OF this] a-unshared
have a ∈ read-only $\mathcal{S}_{sb}$
  by fastforce

from read-only-unacquired-share [OF read-only-unowned [OF i-bound ts$_{sb}$-i]
  weak-sharing-consis [OF i-bound ts$_{sb}$-i] this] a-unowned-sb
have a ∈ read-only (share sb $\mathcal{S}_{sb}$)
  by auto

with a-not-ro **show** False
  by simp
qed

with reads-consistent-mem-eq-on-non-volatile-reads [OF - subset-refl reads-consis-flush-m]
have reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?m-A suspends$_j$
  by (auto simp add: suspends$_j$)

hence reads-consis-m-A-ys: reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?m-A ys
  by (simp add: split-suspends$_j$ reads-consistent-append)

from valid-history [OF j-bound′′ ts_{sb}-j]
have h-consis:
history-consistent j_{sbj} (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof −
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from valid-write-sops [OF j-bound′′ ts_{sb}-j]
have ∀sop∈write-sops (?take-sb_j@?suspends). valid-sop sop
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain valid-sops-take: ∀sop∈write-sops ?take-sb_j. valid-sop sop **and**
valid-sops-drop: ∀sop∈write-sops ys. valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound′′ ts_{sb}-j]
have distinct-read-tmps (?take-sb_j@$suspends_j$)
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb_j ∩ read-tmps $suspends_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps $suspends_j$
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp only: distinct-read-tmps-append)
done

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop h-consis]
last-prog-hd-prog
have hist-consis′: history-consistent j_{sbj} (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis]
have no-vol-read: outstanding-refs is-volatile-$Read_{sb}$ ys = {}
by (auto simp add: outstanding-refs-append $suspends_j$ [symmetric]

split-suspends$_j$)

from flush-store-buffer-append [
OF j-bound'''' is$_j$ [simplified split-suspends$_j$] cph [simplified suspends$_j$]
ts-A-j [simplified split-suspends$_j$] refl lp [simplified split-suspends$_j$] reads-consis-m-A-ys hist-consis' [simplified split-suspends$_j$] valid-sops-drop distinct-read-tmps-drop [simplified split-suspends$_j$]
no-volatile-Read$_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=?share-A]
obtain is$_j$' $\mathcal{R}_j$' **where**
is$_j$': instrs (Read$_{sb}$ False a' t v# zs) @ is$_{sbj}$ =
is$_j$' @ prog-instrs (Read$_{sb}$ False a' t v# zs) **and**
steps-ys: (?ts-A, ?m-A, ?share-A) $\Rightarrow_d{}^*$
(?ts-A[j:= (last-prog (hd-prog p$_j$ (Read$_{sb}$ False a' t v# zs)) ys,
is$_j$',
j$_{sbj}$ |' (dom j$_{sbj}$ − read-tmps (Read$_{sb}$ False a' t v# zs)),(),
$\mathcal{D}_j$ $\vee$ outstanding-refs is-volatile-Write$_{sb}$ ys $\neq$ {}, acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$),$\mathcal{R}_j$')],
flush ys ?m-A,
share ys ?share-A)
(**is** (-,-,-) $\Rightarrow_d{}^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto)

note conflict-computation = rtranclp-trans [OF rtranclp-r-rtranclp [OF steps-flush-sb, OF store-step] steps-ys]
from cph
have causal-program-history is$_{sbj}$ ((ys @ [Read$_{sb}$ False a' t v]) @ zs)
by simp
from causal-program-history-suffix [OF this]
have cph': causal-program-history is$_{sbj}$ zs.
interpret causal$_j$: causal-program-history is$_{sbj}$ zs **by** (rule cph')

from causal$_j$.causal-program-history [of [], simplified, OF refl] is$_j$'
obtain is$_j$''
where is$_j$': is$_j$' = Read False a' t#is$_j$'' **and**
is$_j$'': instrs zs @ is$_{sbj}$ = is$_j$'' @ prog-instrs zs
by clarsimp

from j-bound'''
have j-bound-ys: j < length ?ts-ys
by auto

from j-bound-ys neq-i-j
have ts-ys-j: ?ts-ys!j=(last-prog (hd-prog p$_j$ (Read$_{sb}$ False a' t v# zs)) ys, is$_j$',
j$_{sbj}$ |' (dom j$_{sbj}$ − read-tmps (Read$_{sb}$ False a' t v# zs)),(),
$\mathcal{D}_j$ $\vee$ outstanding-refs is-volatile-Write$_{sb}$ ys $\neq$ {},
acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$),$\mathcal{R}_j$')
by auto

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]

```
  have safe-delayed (?ts-ys,?m-ys,?shared-ys).

  from safe-delayedE [OF this j-bound-ys ts-ys-j, simplified isj']
  have a' ∈ acquired True ys (acquired True ?take-sbj 𝒪j) ∨
          a' ∈ read-only (share ys (share ?drop-sb 𝒮 ⊕W R ⊖A L))
    apply cases
    apply (auto simp add: Let-def issb)
    done
  with a'-unacq
  have a'-ro: a' ∈ read-only (share ys (share ?drop-sb 𝒮 ⊕W R ⊖A L))
    by auto
  from a'-in
  have a'-not-ro: a' ∉ read-only (share ?drop-sb 𝒮 ⊕W R ⊖A L)
    by (auto simp add:  in-read-only-convs)

  have a' ∈ 𝒪j ∪ all-acquired sbj
  proof -
    {
assume a-notin: a' ∉ 𝒪j ∪ all-acquired sbj
from weak-sharing-consis [OF j-bound'' tssb-j]
have weak-sharing-consistent 𝒪j sbj.
with weak-sharing-consistent-append [of 𝒪j ?take-sbj ?drop-sbj]
have weak-sharing-consistent (acquired True ?take-sbj 𝒪j) suspendsj
  by (auto simp add: suspendsj)

with split-suspendsj
have weak-consis: weak-sharing-consistent (acquired True ?take-sbj 𝒪j) ys
  by (simp add: weak-sharing-consistent-append)
from all-acquired-append [of ?take-sbj ?drop-sbj]
have all-acquired ys ⊆ all-acquired sbj
  apply (clarsimp)
  apply (clarsimp simp add: suspendsj [symmetric] split-suspendsj all-acquired-append)
  done

            with a-notin acquired-takeWhile-non-volatile-Writesb [of sbj 𝒪j]
              all-acquired-append [of ?take-sbj ?drop-sbj]
have a' ∉ acquired True (takeWhile (Not ∘ is-volatile-Writesb) sbj) 𝒪j ∪ all-acquired ys
              by auto

from read-only-share-unowned [OF weak-consis this a'-ro]
have a' ∈ read-only (share ?drop-sb 𝒮 ⊕W R ⊖A L) .

with a'-not-ro have False
  by auto
    }
    thus ?thesis by blast
  qed

  moreover
  from A-unaquired-by-others [rule-format, OF j-bound neq-i-j] tssb-j j-bound
```

have A ∩ all-acquired sb_j = {}
  by (auto simp add: Let-def)
moreover
from A-unowned-by-others [rule-format, OF j-bound″ neq-i-j] ts_{sb}-j j-bound
have A ∩ $\mathcal{O}_j$ = {}
  by (auto simp add: Let-def dest: all-shared-acquired-in)
moreover note a′-in
ultimately
show False
  by auto
qed
}
thus ?thesis
  by (auto simp add: Let-def)
qed

have valid-own′: valid-ownership $\mathcal{S}_{sb}$′ ts_{sb}′
proof (intro-locales)
show outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}_{sb}$′ ts_{sb}′
proof −
  from outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ts_{sb}-i]
  have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ (sb @ [$Write_{sb}$ True a (D,f) (f j_{sb}) A L R W])
    by (auto simp add: non-volatile-owned-or-read-only-append)
  from outstanding-non-volatile-refs-owned-or-read-only-nth-update [OF i-bound this]
  show ?thesis **by** (simp add: ts_{sb}′ sb′ $\mathcal{O}_{sb}$′ $\mathcal{S}_{sb}$′)
qed
next
show outstanding-volatile-writes-unowned-by-others ts_{sb}′
proof (unfold-locales)
  fix i_1 j p_1 is_1 $\mathcal{O}_1$ $\mathcal{R}_1$ $\mathcal{D}_1$ xs_1 sb_1 p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs_j sb_j
  assume i_1-bound: i_1 < length ts_{sb}′
  assume j-bound: j < length ts_{sb}′
  assume i_1-j: $i_1 \neq j$
  assume ts-i_1: ts_{sb}′!i_1 = (p_1,is_1,xs_1,sb_1,$\mathcal{D}_1$,$\mathcal{O}_1$,$\mathcal{R}_1$)
  assume ts-j: ts_{sb}′!j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
  show ($\mathcal{O}_j$ ∪ all-acquired sb_j) ∩ outstanding-refs is-volatile-$Write_{sb}$ sb_1 = {}
  proof (cases i_1=i)
    case True
    with i_1-j **have** i-j: i≠j
      by simp

    from j-bound **have** j-bound′: j < length ts_{sb}
      by (simp add: ts_{sb}′)
    hence j-bound″: j < length (map owned ts_{sb})
      by simp
    from ts-j i-j **have** ts-j′: ts_{sb}!j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
      by (simp add: ts_{sb}′)
    from a-unowned-others [rule-format, OF - i-j] i-j ts-j j-bound
    obtain a-notin-j: a ∉ acquired True (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j) $\mathcal{O}_j$ **and**

a-unshared: $a \notin$ all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_j$)
by (auto simp add: Let-def ts$_{sb}$')
from a-not-acquired-others [rule-format, OF - i-j] i-j ts-j j-bound
have a-notin-acq: $a \notin$ all-acquired sb$_j$
by (auto simp add: Let-def ts$_{sb}$')
from outstanding-volatile-writes-unowned-by-others
[OF i-bound j-bound' i-j ts$_{sb}$-i ts-j']
have ($\mathcal{O}_j$ $\cup$ all-acquired sb$_j$) $\cap$ outstanding-refs is-volatile-Write$_{sb}$ sb = {}.
with ts-i$_1$ a-notin-j a-unshared a-notin-acq True i-bound **show** ?thesis
by (auto simp add: ts$_{sb}$' sb' outstanding-refs-append
acquired-takeWhile-non-volatile-Write$_{sb}$ dest: all-shared-acquired-in)
next
case False
note i$_1$-i = this
from i$_1$-bound **have** i$_1$-bound': $i_1 <$ length ts$_{sb}$
by (simp add: ts$_{sb}$')
from ts-i$_1$ False **have** ts-i$_1$': ts$_{sb}$!i$_1$ = (p$_1$,is$_1$,xs$_1$,sb$_1$,$\mathcal{D}_1$,$\mathcal{O}_1$,$\mathcal{R}_1$)
by (simp add: ts$_{sb}$')
show ?thesis
proof (cases j=i)
case True

from i$_1$-bound'
have i$_1$-bound'': $i_1 <$ length (map owned ts$_{sb}$)
by simp

from outstanding-volatile-writes-unowned-by-others
[OF i$_1$-bound' i-bound i$_1$-i ts-i$_1$' ts$_{sb}$-i]
have ($\mathcal{O}_{sb}$ $\cup$ all-acquired sb) $\cap$ outstanding-refs is-volatile-Write$_{sb}$ sb$_1$ = {}.
moreover
from A-unused-by-others [rule-format, OF - False [symmetric]] False ts-i$_1$ i$_1$-bound
have A $\cap$ outstanding-refs is-volatile-Write$_{sb}$ sb$_1$ = {}
by (auto simp add: Let-def ts$_{sb}$')

ultimately
show ?thesis
using ts-j True ts$_{sb}$'
by (auto simp add: i-bound ts$_{sb}$' $\mathcal{O}_{sb}$' sb' all-acquired-append)
next
case False
from j-bound **have** j-bound': $j <$ length ts$_{sb}$
by (simp add: ts$_{sb}$')
from ts-j False **have** ts-j': ts$_{sb}$!j = (p$_j$,is$_j$,xs$_j$,sb$_j$,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (simp add: ts$_{sb}$')
from outstanding-volatile-writes-unowned-by-others
[OF i$_1$-bound' j-bound' i$_1$-j ts-i$_1$' ts-j']
show ($\mathcal{O}_j$ $\cup$ all-acquired sb$_j$) $\cap$ outstanding-refs is-volatile-Write$_{sb}$ sb$_1$ = {} .
qed
qed
qed

next

show ownership-distinct ts_{sb}'

proof –

have $\forall$j<length ts_{sb}. i $\neq$ j $\longrightarrow$

(let (p_j, is_j, j_j, sb_j, $\mathcal{D}_j$, $\mathcal{O}_j$,$\mathcal{R}_j$) = ts_{sb} ! j

in ($\mathcal{O}_{sb}$ $\cup$ all-acquired sb$'$) $\cap$ ($\mathcal{O}_j$ $\cup$ all-acquired sb_j) = {})

proof –

{

fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ acq_j j_j sb_j

assume neq-i-j: i $\neq$ j

assume j-bound: j < length ts_{sb}

assume ts_{sb}-j: ts_{sb} ! j = (p_j, is_j, j_j, sb_j, $\mathcal{D}_j$, $\mathcal{O}_j$,$\mathcal{R}_j$)

have ($\mathcal{O}_{sb}$ $\cup$ all-acquired sb$'$) $\cap$ ($\mathcal{O}_j$ $\cup$ all-acquired sb_j) = {}

proof –

{

fix a$'$

assume a$'$-in-i: a$'$ $\in$ ($\mathcal{O}_{sb}$ $\cup$ all-acquired sb$'$)

assume a$'$-in-j: a$'$ $\in$ ($\mathcal{O}_j$ $\cup$ all-acquired sb_j)

have False

proof –

from a$'$-in-i **have** a$'$ $\in$ ($\mathcal{O}_{sb}$ $\cup$ all-acquired sb) $\vee$ a$'$ $\in$ A

by (simp add: sb$'$ all-acquired-append)

then show False

proof

assume a$'$ $\in$ ($\mathcal{O}_{sb}$ $\cup$ all-acquired sb)

with ownership-distinct [OF i-bound j-bound neq-i-j ts_{sb}-i ts_{sb}-j] a$'$-in-j

show ?thesis

by auto

next

assume a$'$ $\in$ A

moreover

have j-bound$'$: j < length (map owned ts_{sb})

using j-bound **by** auto

from A-unowned-by-others [rule-format, OF - neq-i-j] ts_{sb}-j j-bound

obtain A $\cap$ acquired True (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j) $\mathcal{O}_j$ = {} **and**

A $\cap$ all-shared (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j) = {}

by (auto simp add: Let-def)

moreover

from A-unaquired-by-others [rule-format, OF - neq-i-j] ts_{sb}-j j-bound

have A $\cap$ all-acquired sb_j = {}

by auto

ultimately

show ?thesis

using a$'$-in-j

by (auto dest: all-shared-acquired-in)

qed

qed

}

then show ?thesis **by** auto

qed

}
then show ?thesis **by** (fastforce simp add: Let-def)
qed

from ownership-distinct-nth-update [OF i-bound ts_{sb}-i this]
show ?thesis **by** (simp add: ts_{sb}' $\mathcal{O}_{sb}'$ sb$'$)
qed
next
show read-only-reads-unowned ts_{sb}'
proof
fix n m
fix p_n is_n $\mathcal{O}_n$ $\mathcal{R}_n$ $\mathcal{D}_n$ j_n sb_n p_m is_m $\mathcal{O}_m$ $\mathcal{R}_m$ $\mathcal{D}_m$ j_m sb_m
assume n-bound: n < length ts_{sb}'
and m-bound: m < length ts_{sb}'
and neq-n-m: n≠m
and nth: ts_{sb}'!n = (p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
and mth: ts_{sb}'!m =(p_m, is_m, j_m, sb_m, $\mathcal{D}_m$, $\mathcal{O}_m$,$\mathcal{R}_m$)
from n-bound **have** n-bound$'$: n < length ts_{sb} **by** (simp add: ts_{sb}')
from m-bound **have** m-bound$'$: m < length ts_{sb} **by** (simp add: ts_{sb}')

show ($\mathcal{O}_m$ ∪ all-acquired sb_m) ∩
read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n) $\mathcal{O}_n$)
(dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n) =
{}
proof (cases m=i)
case True
with neq-n-m **have** neq-n-i: n≠i
by auto
with n-bound nth i-bound **have** nth$'$: ts_{sb}!n =(p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
by (auto simp add: ts_{sb}')
note read-only-reads-unowned [OF n-bound$'$ i-bound neq-n-i nth$'$ ts_{sb}-i]
moreover
from A-no-read-only-reads-by-others [rule-format, OF - neq-n-i [symmetric]] n-bound$'$ nth$'$
have A ∩ read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n) $\mathcal{O}_n$)
(dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n) =
{}
by auto
ultimately
show ?thesis
using True ts_{sb}-i nth$'$ mth n-bound$'$ m-bound$'$
by (auto simp add: ts_{sb}' $\mathcal{O}_{sb}'$ sb$'$ all-acquired-append)
next
case False
note neq-m-i = this
with m-bound mth i-bound **have** mth$'$: ts_{sb}!m = (p_m, is_m, j_m, sb_m, $\mathcal{D}_m$, $\mathcal{O}_m$,$\mathcal{R}_m$)
by (auto simp add: ts_{sb}')
show ?thesis
proof (cases n=i)

case True
note read-only-reads-unowned [OF i-bound m-bound′ neq-m-i [symmetric] ts_{sb}-i mth′]
then show ?thesis
using True neq-m-i ts_{sb}-i nth mth n-bound′ m-bound′
apply (case-tac outstanding-refs (is-volatile-$Write_{sb}$) sb = {})
apply (clarsimp simp add: outstanding-vol-write-take-drop-appends
acquired-append read-only-reads-append ts_{sb}′ sb′ $\mathcal{O}_{sb}$′)+
done
next
case False
with n-bound nth i-bound **have** nth′: ts_{sb}!n =(p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
by (auto simp add: ts_{sb}′)
from read-only-reads-unowned [OF n-bound′ m-bound′ neq-n-m nth′ mth′] False neq-m-i
show ?thesis
by (clarsimp)
qed
qed
qed
qed

have valid-hist′: valid-history program-step ts_{sb}′
proof −
from valid-history [OF i-bound ts_{sb}-i]
have history-consistent j_{sb} (hd-prog p_{sb} sb) sb**.**
with valid-write-sops [OF i-bound ts_{sb}-i] D-subset
valid-implies-valid-prog-hd [OF i-bound ts_{sb}-i valid]
have history-consistent j_{sb} (hd-prog p_{sb} (sb@[$Write_{sb}$ True a (D,f) (f j_{sb}) A L R W]))
(sb@ [$Write_{sb}$ True a (D,f) (f j_{sb}) A L R W])
apply −
apply (rule history-consistent-appendI)
apply (auto simp add: hd-prog-append-$Write_{sb}$)
done
from valid-history-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}′ sb′ j_{sb}′)
qed

have valid-reads′: valid-reads m_{sb} ts_{sb}′
proof −
from valid-reads [OF i-bound ts_{sb}-i]
have reads-consistent False $\mathcal{O}_{sb}$ m_{sb} sb **.**
from reads-consistent-snoc-$Write_{sb}$ [OF this]
have reads-consistent False $\mathcal{O}_{sb}$ m_{sb} (sb @ [$Write_{sb}$ True a (D,f) (f j_{sb}) A L R W])**.**
from valid-reads-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}′ sb′ $\mathcal{O}_{sb}$′)
qed

have valid-sharing′: valid-sharing $\mathcal{S}_{sb}$′ ts_{sb}′
proof (intro-locales)

from outstanding-non-volatile-writes-unshared [OF i-bound ts_{sb}-i]
have non-volatile-writes-unshared $\mathcal{S}_{sb}$ (sb @ [$Write_{sb}$ True a (D,f) (f j_{sb}) A L R W])
  by (auto simp add: non-volatile-writes-unshared-append)
from outstanding-non-volatile-writes-unshared-nth-update [OF i-bound this]
show outstanding-non-volatile-writes-unshared $\mathcal{S}_{sb}'$ ts_{sb}'
  by (simp add: ts_{sb}' sb$'$ $\mathcal{S}_{sb}'$)
    next
from sharing-consis [OF i-bound ts_{sb}-i]
have consis$'$: sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ sb**.**
from A-shared-owned
have A $\subseteq$ dom (share ?drop-sb $\mathcal{S}$) $\cup$ acquired True sb $\mathcal{O}_{sb}$
  by (simp add: sharing-consistent-append acquired-takeWhile-non-volatile-$Write_{sb}$)
moreover have dom (share ?drop-sb $\mathcal{S}$) $\subseteq$ dom $\mathcal{S}$ $\cup$ dom (share sb $\mathcal{S}_{sb}$)
proof
  fix a$'$
  assume a$'$-in: a$'$ $\in$ dom (share ?drop-sb $\mathcal{S}$)
  from share-unshared-in [OF a$'$-in]
  show a$'$ $\in$ dom $\mathcal{S}$ $\cup$ dom (share sb $\mathcal{S}_{sb}$)
  proof
    assume a$'$ $\in$ dom (share ?drop-sb Map.empty)
    from share-mono-in [OF this] share-append [of ?take-sb ?drop-sb]
    have a$'$ $\in$ dom (share sb $\mathcal{S}_{sb}$)
      by auto
    thus ?thesis
      by simp
  next
    assume a$'$ $\in$ dom $\mathcal{S}$ $\wedge$ a$'$ $\notin$ all-unshared ?drop-sb
    thus ?thesis **by** auto
  qed
qed
ultimately
have A-subset: A $\subseteq$ dom $\mathcal{S}$ $\cup$ dom (share sb $\mathcal{S}_{sb}$) $\cup$ acquired True sb $\mathcal{O}_{sb}$
  by auto

with A-unowned-by-others

have A $\subseteq$ dom (share sb $\mathcal{S}_{sb}$) $\cup$ acquired True sb $\mathcal{O}_{sb}$
proof –
  {
    fix x
    assume x-A: x $\in$ A
    have x $\in$ dom (share sb $\mathcal{S}_{sb}$) $\cup$ acquired True sb $\mathcal{O}_{sb}$
    proof –
      {
        assume x $\in$ dom $\mathcal{S}$

from share-all-until-volatile-write-share-acquired [OF ‹sharing-consis $\mathcal{S}_{sb}$ ts_{sb}›

i-bound ts_{sb}-i this [simplified $\mathcal{S}$]]
A-unowned-by-others x-A

have ?thesis
by (fastforce simp add: Let-def)
}
with A-subset **show** ?thesis **using** x-A **by** auto
qed
}
thus ?thesis **by** blast
qed
with consis′ L-subset A-R R-acq
have sharing-consistent $\mathcal{S}_{\mathsf{sb}}$ $\mathcal{O}_{\mathsf{sb}}$ (sb @ [Write$_{\mathsf{sb}}$ True a (D,f) (f j$_{\mathsf{sb}}$) A L R W])
by (simp add: sharing-consistent-append acquired-takeWhile-non-volatile-Write$_{\mathsf{sb}}$)
from sharing-consis-nth-update [OF i-bound this]
show sharing-consis $\mathcal{S}_{\mathsf{sb}}'$ ts$_{\mathsf{sb}}'$
by (simp add: ts$_{\mathsf{sb}}'$ $\mathcal{O}_{\mathsf{sb}}'$ sb′ $\mathcal{S}_{\mathsf{sb}}'$)
next
from read-only-unowned-nth-update [OF i-bound read-only-unowned [OF i-bound ts$_{\mathsf{sb}}$-i]
]
show read-only-unowned $\mathcal{S}_{\mathsf{sb}}'$ ts$_{\mathsf{sb}}'$
by (simp add: $\mathcal{S}_{\mathsf{sb}}'$ ts$_{\mathsf{sb}}'$ $\mathcal{O}_{\mathsf{sb}}'$)
next
from unowned-shared-nth-update [OF i-bound ts$_{\mathsf{sb}}$-i subset-refl]
show unowned-shared $\mathcal{S}_{\mathsf{sb}}'$ ts$_{\mathsf{sb}}'$
by (simp add: ts$_{\mathsf{sb}}'$ sb′ $\mathcal{O}_{\mathsf{sb}}'$ $\mathcal{S}_{\mathsf{sb}}'$)
next
from a-not-ro no-outstanding-write-to-read-only-memory [OF i-bound ts$_{\mathsf{sb}}$-i]
have no-write-to-read-only-memory $\mathcal{S}_{\mathsf{sb}}$ (sb @ [Write$_{\mathsf{sb}}$ True a (D,f) (f j$_{\mathsf{sb}}$) A L R W])
by (simp add: no-write-to-read-only-memory-append)

from no-outstanding-write-to-read-only-memory-nth-update [OF i-bound this]
show no-outstanding-write-to-read-only-memory $\mathcal{S}_{\mathsf{sb}}'$ ts$_{\mathsf{sb}}'$
by (simp add: $\mathcal{S}_{\mathsf{sb}}'$ ts$_{\mathsf{sb}}'$ sb′)
qed

have tmps-distinct′: tmps-distinct ts$_{\mathsf{sb}}'$
proof (intro-locales)
from load-tmps-distinct [OF i-bound ts$_{\mathsf{sb}}$-i]
have distinct-load-tmps is$_{\mathsf{sb}}'$ **by** (simp add: is$_{\mathsf{sb}}$)
from load-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-distinct ts$_{\mathsf{sb}}'$ **by** (simp add: ts$_{\mathsf{sb}}'$)
next
from read-tmps-distinct [OF i-bound ts$_{\mathsf{sb}}$-i]
have distinct-read-tmps (sb @ [Write$_{\mathsf{sb}}$ True a (D, f) (f j$_{\mathsf{sb}}$) A L R W])
by (auto simp add: distinct-read-tmps-append)
from read-tmps-distinct-nth-update [OF i-bound this]
show read-tmps-distinct ts$_{\mathsf{sb}}'$ **by** (simp add: ts$_{\mathsf{sb}}'$ sb′)
next
from load-tmps-read-tmps-distinct [OF i-bound ts$_{\mathsf{sb}}$-i]
have load-tmps is$_{\mathsf{sb}}'$ ∩ read-tmps (sb @ [Write$_{\mathsf{sb}}$ True a (D, f) (f j$_{\mathsf{sb}}$) A L R W]) ={}
by (auto simp add: read-tmps-append is$_{\mathsf{sb}}$)
from load-tmps-read-tmps-distinct-nth-update [OF i-bound this]

show load-tmps-read-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}' sb′)
qed

have valid-sops′: valid-sops ts_{sb}'
proof −
from valid-store-sops [OF i-bound ts_{sb}-i]
obtain valid-Df: valid-sop (D,f) **and**
valid-store-sops′: ∀sop∈store-sops is_{sb}'. valid-sop sop
by (auto simp add: is_{sb})
from valid-Df valid-write-sops [OF i-bound ts_{sb}-i]
have valid-write-sops′: ∀sop∈write-sops (sb@ [$Write_{sb}$ True a (D, f) (f j_{sb}) A L R W]).
valid-sop sop
by (auto simp add: write-sops-append)
from valid-sops-nth-update [OF i-bound valid-write-sops′ valid-store-sops′]
show ?thesis **by** (simp add: ts_{sb}' sb′)
qed

have valid-dd′: valid-data-dependency ts_{sb}'
proof −
from data-dependency-consistent-instrs [OF i-bound ts_{sb}-i]
obtain D-indep: D ∩ load-tmps is_{sb}' = {} **and**
dd-is: data-dependency-consistent-instrs (dom j_{sb}') is_{sb}'
by (auto simp add: is_{sb} j_{sb}')
from load-tmps-write-tmps-distinct [OF i-bound ts_{sb}-i] D-indep
have load-tmps is_{sb}' ∩ ⋃(fst ‘ write-sops (sb@ [$Write_{sb}$ True a (D, f) (f j_{sb}) A L R W]))
={}
by (auto simp add: write-sops-append is_{sb})
from valid-data-dependency-nth-update [OF i-bound dd-is this]
show ?thesis **by** (simp add: ts_{sb}' sb′)
qed

have load-tmps-fresh′: load-tmps-fresh ts_{sb}'
proof −
from load-tmps-fresh [OF i-bound ts_{sb}-i]
have load-tmps is_{sb}' ∩ dom j_{sb} = {}
by (auto simp add: is_{sb})
from load-tmps-fresh-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' j_{sb}')
qed

have enough-flushs′: enough-flushs ts_{sb}'
proof −
from clean-no-outstanding-volatile-$Write_{sb}$ [OF i-bound ts_{sb}-i]
have ¬ True ⟶ outstanding-refs is-volatile-$Write_{sb}$ (sb@[$Write_{sb}$ True a (D,f) (f j_{sb}) A
L R W]) = {}
by (auto simp add: outstanding-refs-append)
from enough-flushs-nth-update [OF i-bound this]
show ?thesis
by (simp add: ts_{sb}' sb′ $\mathcal{D}_{sb}'$)
qed

have valid-program-history′: valid-program-history ts_{sb}′
proof −
from valid-program-history [OF i-bound ts_{sb}-i]
have causal-program-history is_{sb} sb .
then have causal′: causal-program-history is_{sb}′ (sb@[$Write_{sb}$ True a (D,f) (f j_{sb}) A L R W])
by (auto simp: causal-program-history-Write is_{sb})
from valid-last-prog [OF i-bound ts_{sb}-i]
have last-prog p_{sb} sb = p_{sb}.
hence last-prog p_{sb} (sb @ [$Write_{sb}$ True a (D,f) (f j_{sb}) A L R W]) = p_{sb}
by (simp add: last-prog-append-$Write_{sb}$)
from valid-program-history-nth-update [OF i-bound causal′ this]
show ?thesis
by (simp add: ts_{sb}′ sb′)
qed

show ?thesis
proof (cases outstanding-refs is-volatile-$Write_{sb}$ sb = {})
case True

from True **have** flush-all: takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb = sb
by (auto simp add: outstanding-refs-conv)

from True **have** suspend-nothing: dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb = []
by (auto simp add: outstanding-refs-conv)

hence suspends-empty: suspends = []
by (simp add: suspends)

from suspends-empty is-sim **have** is: is = Write True a (D,f) A L R W# is_{sb}′
by (simp add: is_{sb})
with suspends-empty ts-i
have ts-i: ts!i = (p_{sb}, Write True a (D,f) A L R W# is_{sb}′, j_{sb},(),$\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$, release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
by simp

have (ts, m, $\mathcal{S}$) $\Rightarrow_d^*$ (ts, m, $\mathcal{S}$) **by** auto

moreover

note flush-commute =
flush-all-until-volatile-write-append-volatile-write-commute
[OF True i-bound ts_{sb}-i]

from True
have drop-app: dropWhile (Not ∘ is-volatile-$Write_{sb}$)
(sb@[$Write_{sb}$ True a (D,f) (f j_{sb}) A L R W]) =
[$Write_{sb}$ True a (D,f) (f j_{sb}) A L R W]

by (auto simp add: outstanding-refs-conv)

have (ts_{sb}',m_{sb},$\mathcal{S}_{sb}'$) $\sim$ (ts,m,$\mathcal{S}$)
apply (rule sim-config.intros)
apply (simp add: m flush-commute ts_{sb}' j_{sb}' $\mathcal{O}_{sb}'$ $\mathcal{R}_{sb}'$ sb')
using share-all-until-volatile-write-Write-commute
[OF i-bound ts_{sb}-i [simplified is_{sb}]]
apply (clarsimp simp add: $\mathcal{S}$ $\mathcal{S}_{sb}'$ ts_{sb}' sb' $\mathcal{O}_{sb}'$ $\mathcal{R}_{sb}'$ j_{sb}')
using leq
apply (simp add: ts_{sb}')
using i-bound i-bound' ts-sim ts-i
apply (clarsimp simp add: Let-def nth-list-update drop-app
ts_{sb}' sb' $\mathcal{O}_{sb}'$ $\mathcal{R}_{sb}'$ $\mathcal{S}_{sb}'$ j_{sb}' $\mathcal{D}_{sb}'$ outstanding-refs-append takeWhile-tail flush-all
split: if-split-asm)
done

ultimately show ?thesis
using valid-own' valid-hist' valid-reads' valid-sharing' tmps-distinct'
valid-sops'
valid-dd' load-tmps-fresh' enough-flushs'
valid-program-history' valid' m_{sb}' $\mathcal{S}_{sb}'$
by auto
next
case False

then obtain r **where** r-in: r $\in$ set sb **and** volatile-r: is-volatile-$Write_{sb}$ r
by (auto simp add: outstanding-refs-conv)
from takeWhile-dropWhile-real-prefix
[OF r-in, of (Not $\circ$ is-volatile-$Write_{sb}$), simplified, OF volatile-r]
obtain a' v' sb'' A'' L'' R'' W'' sop' **where**
sb-split: sb = takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb @ $Write_{sb}$ True a' sop' v' A'' L'' R'' W''# sb''
and
drop: dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb = $Write_{sb}$ True a' sop' v' A'' L'' R'' W''# sb''
apply (auto)
subgoal for y ys
apply (case-tac y)
apply auto
done
done
from drop suspends **have** suspends: suspends = $Write_{sb}$ True a' sop' v' A'' L'' R'' W''# sb''
by simp

have (ts, m, $\mathcal{S}$) $\Rightarrow_{d}^{*}$ (ts, m, $\mathcal{S}$) **by** auto

moreover

note flush-commute =

flush-all-until-volatile-write-append-unflushed [OF False i-bound ts_{sb}-i]

have $Write_{sb}$ True a′ sop′ v′ A″ L″ R″ W″ ∈ set sb
by (subst sb-split) auto
note drop-app = dropWhile-append1
[OF this, of (Not ∘ is-volatile-$Write_{sb}$), simplified]

have (ts_{sb}′,m_{sb},$\mathcal{S}_{sb}$′) ∼ (ts,m,$\mathcal{S}$)
apply (rule sim-config.intros)
apply (simp add: m flush-commute ts_{sb}′ $\mathcal{O}_{sb}$′ $\mathcal{R}_{sb}$′ j_{sb}′ sb′)
using share-all-until-volatile-write-Write-commute
[OF i-bound ts_{sb}-i [simplified is_{sb}]]
apply (clarsimp simp add: $\mathcal{S}$ $\mathcal{S}_{sb}$′ ts_{sb}′ sb′ $\mathcal{O}_{sb}$′ $\mathcal{R}_{sb}$′ j_{sb}′)
using leq
apply (simp add: ts_{sb}′)
using i-bound i-bound′ ts-sim ts-i is-sim
apply (clarsimp simp add: Let-def nth-list-update is-sim drop-app
read-tmps-append suspends
prog-instrs-append-$Write_{sb}$ instrs-append-$Write_{sb}$ hd-prog-append-$Write_{sb}$
drop is_{sb} ts_{sb}′ sb′ $\mathcal{O}_{sb}$′ $\mathcal{S}_{sb}$′ $\mathcal{R}_{sb}$′ j_{sb}′ $\mathcal{D}_{sb}$′ outstanding-refs-append takeWhile-tail
release-append split: if-split-asm)
done
ultimately show ?thesis
using valid-own′ valid-hist′ valid-reads′ valid-sharing′ tmps-distinct′ valid-dd′
valid-sops′ load-tmps-fresh′ enough-flushs′
valid-program-history′ valid′ m_{sb}′ $\mathcal{S}_{sb}$′
by (auto simp del: fun-upd-apply)
qed
next
case SBHFence
then obtain
is_{sb}: is_{sb} = Fence # is_{sb}′ **and**
sb: sb=[] **and**
$\mathcal{O}_{sb}$′: $\mathcal{O}_{sb}$′=$\mathcal{O}_{sb}$ **and**
$\mathcal{R}_{sb}$′: $\mathcal{R}_{sb}$′=Map.empty **and**
j_{sb}′: j_{sb}′ = j_{sb} **and**
$\mathcal{D}_{sb}$′: ¬ $\mathcal{D}_{sb}$′ **and**
sb′: sb′=sb **and**
m_{sb}′: m_{sb}′ = m_{sb} **and**
$\mathcal{S}_{sb}$′: $\mathcal{S}_{sb}$′=$\mathcal{S}_{sb}$
by auto

have valid-own′: valid-ownership $\mathcal{S}_{sb}$′ ts_{sb}′
proof (intro-locales)
show outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}_{sb}$′ ts_{sb}′
proof −
have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ []
by simp
from outstanding-non-volatile-refs-owned-or-read-only-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}′ sb′ sb $\mathcal{O}_{sb}$′ $\mathcal{S}_{sb}$′)

qed
next
from outstanding-volatile-writes-unowned-by-others-store-buffer
[OF i-bound ts_{sb}-i subset-refl]
show outstanding-volatile-writes-unowned-by-others ts_{sb}'
by (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$)
next
from read-only-reads-unowned-nth-update [OF i-bound ts_{sb}-i, of [] $\mathcal{O}_{sb}$]
show read-only-reads-unowned ts_{sb}'
by (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$)
next
from ownership-distinct-instructions-read-value-store-buffer-independent
[OF i-bound ts_{sb}-i]
show ownership-distinct ts_{sb}'
by (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$)
qed

have valid-hist′: valid-history program-step ts_{sb}'
proof −
from valid-history [OF i-bound ts_{sb}-i]
have history-consistent j_{sb} (hd-prog p_{sb} []) [] **by** simp
from valid-history-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$ j_{sb}')
qed

have valid-reads′: valid-reads m_{sb} ts_{sb}'
proof −
have reads-consistent False $\mathcal{O}_{sb}$ m_{sb} [] **by** simp
from valid-reads-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$)
qed

have valid-sharing′: valid-sharing $\mathcal{S}_{sb}'$ ts_{sb}'
proof (intro-locales)
have non-volatile-writes-unshared $\mathcal{S}_{sb}$ []
by (simp add: sb)
from outstanding-non-volatile-writes-unshared-nth-update [OF i-bound this]
show outstanding-non-volatile-writes-unshared $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: ts_{sb}' sb sb' $\mathcal{S}_{sb}'$)
next
have sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ [] **by** simp
from sharing-consis-nth-update [OF i-bound this]
show sharing-consis $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: ts_{sb}' $\mathcal{O}_{sb}'$ sb' sb $\mathcal{S}_{sb}'$)
next
from read-only-unowned-nth-update [OF i-bound read-only-unowned [OF i-bound ts_{sb}-i]
]
show read-only-unowned $\mathcal{S}_{sb}'$ ts_{sb}'

by (simp add: $\mathcal{S}_{\mathsf{sb}}'$ ts_{sb}' $\mathcal{O}_{\mathsf{sb}}'$)
next
from unowned-shared-nth-update [OF i-bound ts_{sb}-i subset-refl]
show unowned-shared $\mathcal{S}_{\mathsf{sb}}'$ ts_{sb}' **by** (simp add: ts_{sb}' sb$'$ sb $\mathcal{O}_{\mathsf{sb}}'$ $\mathcal{S}_{\mathsf{sb}}'$)
next
from no-outstanding-write-to-read-only-memory-nth-update [OF i-bound, of []]
show no-outstanding-write-to-read-only-memory $\mathcal{S}_{\mathsf{sb}}'$ ts_{sb}'
by (simp add: $\mathcal{S}_{\mathsf{sb}}'$ ts_{sb}' sb$'$ sb)
qed

have tmps-distinct$'$: tmps-distinct ts_{sb}'
proof (intro-locales)
from load-tmps-distinct [OF i-bound ts_{sb}-i]
have distinct-load-tmps is_{sb}'
by (auto simp add: is_{sb} split: instr.splits)
from load-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}' sb$'$ sb $\mathcal{O}_{\mathsf{sb}}'$ is_{sb})
next
from read-tmps-distinct [OF i-bound ts_{sb}-i]
have distinct-read-tmps [] **by** (simp add: ts_{sb}' sb$'$ sb $\mathcal{O}_{\mathsf{sb}}'$)
from read-tmps-distinct-nth-update [OF i-bound this]
show read-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}' sb$'$ sb $\mathcal{O}_{\mathsf{sb}}'$)
next
from load-tmps-read-tmps-distinct [OF i-bound ts_{sb}-i]
load-tmps-distinct [OF i-bound ts_{sb}-i]
have load-tmps is_{sb}' $\cap$ read-tmps [] = {}
by (clarsimp)
from load-tmps-read-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-read-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}' sb$'$ sb $\mathcal{O}_{\mathsf{sb}}'$)
qed

have valid-sops$'$: valid-sops ts_{sb}'
proof $-$
from valid-store-sops [OF i-bound ts_{sb}-i]
obtain
valid-store-sops$'$: $\forall$ sop$\in$store-sops is_{sb}'. valid-sop sop
by (auto simp add: is_{sb} ts_{sb}' sb$'$ sb $\mathcal{O}_{\mathsf{sb}}'$)

from valid-sops-nth-update [OF i-bound - valid-store-sops$'$, **where** sb= []]
show ?thesis **by** (auto simp add: ts_{sb}' sb$'$ sb $\mathcal{O}_{\mathsf{sb}}'$)
qed

have valid-dd$'$: valid-data-dependency ts_{sb}'
proof $-$
from data-dependency-consistent-instrs [OF i-bound ts_{sb}-i]
obtain
dd-is: data-dependency-consistent-instrs (dom j_{sb}') is_{sb}'
by (auto simp add: is_{sb} j_{sb}')
from load-tmps-write-tmps-distinct [OF i-bound ts_{sb}-i]
have load-tmps is_{sb}' $\cap$ $\bigcup$(fst ` write-sops []) = {}

by (auto simp add: write-sops-append)
from valid-data-dependency-nth-update [OF i-bound dd-is this]
show ?thesis **by** (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$)
qed

have load-tmps-fresh$'$: load-tmps-fresh ts_{sb}'
proof −
from load-tmps-fresh [OF i-bound ts_{sb}-i]
have load-tmps is_{sb}' $\cap$ dom j_{sb} = {}
by (auto simp add: is_{sb})
from load-tmps-fresh-nth-update [OF i-bound this]
show ?thesis **by** (simp add: is_{sb} ts_{sb}' sb' sb j_{sb}')
qed

from enough-flushs-nth-update [OF i-bound, **where** sb=[]]
have enough-flushs$'$: enough-flushs ts_{sb}'
by (auto simp add: ts_{sb}' sb' sb)

have valid-program-history$'$: valid-program-history ts_{sb}'
proof −
have causal$'$: causal-program-history is_{sb}' sb'
by (simp add: is_{sb} sb sb')
have last-prog p_{sb} sb' = p_{sb}
by (simp add: sb' sb)
from valid-program-history-nth-update [OF i-bound causal$'$ this]
show ?thesis
by (simp add: ts_{sb}' sb')
qed

from is-sim **have** is: is = Fence # is_{sb}'
by (simp add: suspends sb is_{sb})
with ts-i
have ts-i: ts!i = (p_{sb}, Fence # is_{sb}', j_{sb},(), $\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$, release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
by (simp add: suspends sb)

from direct-memop-step.Fence
have (Fence # is_{sb}',
j_{sb}, (),m,$\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$, release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$, $\mathcal{S}$) $\rightarrow$
(is_{sb}', j_{sb}, (), m, False, acquired True ?take-sb $\mathcal{O}_{sb}$, Map.empty, $\mathcal{S}$).
from direct-computation.concurrent-step.Memop [OF i-bound$'$ ts-i this]
have (ts, m, $\mathcal{S}$) $\Rightarrow_d$
(ts[i := (p_{sb}, is_{sb}', j_{sb}, (), False, acquired True ?take-sb $\mathcal{O}_{sb}$,Map.empty)], m, $\mathcal{S}$).

moreover

have (ts_{sb}',m_{sb},$\mathcal{S}_{sb}'$) $\sim$ (ts[i := (p_{sb},is_{sb}', j_{sb},(), False,acquired True ?take-sb $\mathcal{O}_{sb}$,Map.empty)],m,$\mathcal{S}$)
apply (rule sim-config.intros)

apply (simp add: ts_{sb}' sb' $\mathcal{O}_{sb}'$ $\mathcal{R}_{sb}'$ $\mathcal{S}_{sb}'$ m
flush-all-until-volatile-nth-update-unused [OF i-bound ts_{sb}-i])
using share-all-until-volatile-write-Fence-commute
[OF i-bound ts_{sb}-i [simplified is_{sb} sb]]
apply (clarsimp simp add: $\mathcal{S}$ ts_{sb}' $\mathcal{S}_{sb}'$ is_{sb} $\mathcal{O}_{sb}'$ $\mathcal{R}_{sb}'$ j_{sb}' sb' sb)
using leq
apply (simp add: ts_{sb}')
using i-bound i-bound$'$ ts-sim
apply (clarsimp simp add: Let-def nth-list-update
ts_{sb}' sb' sb $\mathcal{O}_{sb}'$ $\mathcal{R}_{sb}'$ $\mathcal{S}_{sb}'$ $\mathcal{D}_{sb}'$ ex-not j_{sb}'
split: if-split-asm)
done
ultimately
show ?thesis
using valid-own$'$ valid-hist$'$ valid-reads$'$ valid-sharing$'$ tmps-distinct$'$ valid-sops$'$
valid-dd$'$ load-tmps-fresh$'$ enough-flushs$'$
valid-program-history$'$ valid$'$ m_{sb}' $\mathcal{S}_{sb}'$
by (auto simp del: fun-upd-apply)
next
case (SBHRMWReadOnly cond t a D f ret A L R W)
then obtain
is_{sb}: is_{sb} = RMW a t (D,f) cond ret A L R W # is_{sb}' **and**
cond: $\neg$ (cond (j_{sb}(t$\mapsto m_{sb}$ a))) **and**
$\mathcal{O}_{sb}'$: $\mathcal{O}_{sb}'$=$\mathcal{O}_{sb}$ **and**
$\mathcal{R}_{sb}'$: $\mathcal{R}_{sb}'$=Map.empty **and**
j_{sb}': j_{sb}' = j_{sb}(t$\mapsto m_{sb}$ a) **and**
$\mathcal{D}_{sb}'$: $\neg$ $\mathcal{D}_{sb}'$ **and**
sb: sb=[] **and**
sb': sb'=[] **and**
m_{sb}': m_{sb}' = m_{sb} **and**
$\mathcal{S}_{sb}'$: $\mathcal{S}_{sb}'$=$\mathcal{S}_{sb}$
by auto

from safe-RMW-common [OF safe-memop-flush-sb [simplified is_{sb}]]
obtain access-cond: a $\in$ $\mathcal{O}_{sb}$ $\vee$ a $\in$ dom $\mathcal{S}$ **and**
rels-cond: $\forall$j < length ts. i$\neq$j $\longrightarrow$ released (ts!j) a $\neq$ Some False
by (auto simp add: $\mathcal{S}$ sb)

have valid-own$'$: valid-ownership $\mathcal{S}_{sb}'$ ts_{sb}'
proof (intro-locales)
show outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}_{sb}'$ ts_{sb}'
proof −
have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ []
by simp
from outstanding-non-volatile-refs-owned-or-read-only-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$ $\mathcal{S}_{sb}'$)
qed
next
from outstanding-volatile-writes-unowned-by-others-store-buffer

[OF i-bound ts_{sb}-i subset-refl]
show outstanding-volatile-writes-unowned-by-others ts_{sb}'
  by (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$ $\mathcal{S}_{sb}'$)
    next
from read-only-reads-unowned-nth-update [OF i-bound ts_{sb}-i, of [] $\mathcal{O}_{sb}$]
show read-only-reads-unowned ts_{sb}'
  by (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$)
    next
from ownership-distinct-instructions-read-value-store-buffer-independent
[OF i-bound ts_{sb}-i]
show ownership-distinct ts_{sb}'
  by (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$)
    qed

    have valid-hist': valid-history program-step ts_{sb}'
    proof −
from valid-history [OF i-bound ts_{sb}-i]
have history-consistent (j_{sb}(t↦m_{sb} a)) (hd-prog p_{sb} []) [] **by** simp
from valid-history-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$ j_{sb}')
    qed

    have valid-reads': valid-reads m_{sb} ts_{sb}'
    proof −
have reads-consistent False $\mathcal{O}_{sb}$ m_{sb} [] **by** simp
from valid-reads-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$)
    qed

    have valid-sharing': valid-sharing $\mathcal{S}_{sb}'$ ts_{sb}'
    proof (intro-locales)
from outstanding-non-volatile-writes-unshared [OF i-bound ts_{sb}-i]
have non-volatile-writes-unshared $\mathcal{S}_{sb}$ []
  by (simp add: sb)
from outstanding-non-volatile-writes-unshared-nth-update [OF i-bound this]
show outstanding-non-volatile-writes-unshared $\mathcal{S}_{sb}'$ ts_{sb}'
  by (simp add: ts_{sb}' sb sb' $\mathcal{S}_{sb}'$)
    next
have sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ [] **by** simp
from sharing-consis-nth-update [OF i-bound this]
show sharing-consis $\mathcal{S}_{sb}'$ ts_{sb}'
  by (simp add: ts_{sb}' $\mathcal{O}_{sb}'$ sb' sb $\mathcal{S}_{sb}'$)
    next
from read-only-unowned-nth-update [OF i-bound read-only-unowned [OF i-bound ts_{sb}-i]
]
show read-only-unowned $\mathcal{S}_{sb}'$ ts_{sb}'
  by (simp add: $\mathcal{S}_{sb}'$ ts_{sb}' $\mathcal{O}_{sb}'$)
    next

from unowned-shared-nth-update [OF i-bound ts_{sb}-i subset-refl]
show unowned-shared $\mathcal{S}_{sb}'$ ts_{sb}' **by** (simp add: ts_{sb}' sb$'$ sb $\mathcal{O}_{sb}'$ $\mathcal{S}_{sb}'$)
next
from no-outstanding-write-to-read-only-memory-nth-update [OF i-bound, of []]
show no-outstanding-write-to-read-only-memory $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: $\mathcal{S}_{sb}'$ ts_{sb}' sb$'$ sb)
qed

have tmps-distinct$'$: tmps-distinct ts_{sb}'
proof (intro-locales)
from load-tmps-distinct [OF i-bound ts_{sb}-i]
have distinct-load-tmps is_{sb}'
by (auto simp add: is_{sb} split: instr.splits)
from load-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}' sb$'$ sb $\mathcal{O}_{sb}'$ is_{sb})
next
from read-tmps-distinct [OF i-bound ts_{sb}-i]
have distinct-read-tmps [] **by** (simp add: ts_{sb}' sb$'$ sb $\mathcal{O}_{sb}'$)
from read-tmps-distinct-nth-update [OF i-bound this]
show read-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}' sb$'$ sb $\mathcal{O}_{sb}'$)
next
from load-tmps-read-tmps-distinct [OF i-bound ts_{sb}-i]
load-tmps-distinct [OF i-bound ts_{sb}-i]
have load-tmps is_{sb}' $\cap$ read-tmps [] = {}
by (clarsimp)
from load-tmps-read-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-read-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}' sb$'$ sb $\mathcal{O}_{sb}'$)
qed

have valid-sops$'$: valid-sops ts_{sb}'
proof –
from valid-store-sops [OF i-bound ts_{sb}-i]
obtain
valid-store-sops$'$: $\forall$sop$\in$store-sops is_{sb}'. valid-sop sop
by (auto simp add: is_{sb} ts_{sb}' sb$'$ sb $\mathcal{O}_{sb}'$)

from valid-sops-nth-update [OF i-bound - valid-store-sops$'$, **where** sb= []]
show ?thesis **by** (auto simp add: ts_{sb}' sb$'$ sb $\mathcal{O}_{sb}'$)
qed

have valid-dd$'$: valid-data-dependency ts_{sb}'
proof –
from data-dependency-consistent-instrs [OF i-bound ts_{sb}-i]
obtain
dd-is: data-dependency-consistent-instrs (dom j_{sb}') is_{sb}'
by (auto simp add: is_{sb} j_{sb}')
from load-tmps-write-tmps-distinct [OF i-bound ts_{sb}-i]
have load-tmps is_{sb}' $\cap$ $\bigcup$(fst ‘ write-sops []) = {}
by (auto simp add: write-sops-append)
from valid-data-dependency-nth-update [OF i-bound dd-is this]

show ?thesis **by** (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$)
qed

have load-tmps-fresh$'$: load-tmps-fresh ts_{sb}'
proof –
from load-tmps-fresh [OF i-bound ts_{sb}-i]
have load-tmps (RMW a t (D,f) cond ret A L R W# is_{sb}') $\cap$ dom j_{sb} = {}
by (simp add: is_{sb})
moreover
from load-tmps-distinct [OF i-bound ts_{sb}-i] **have** t $\notin$ load-tmps is_{sb}'
by (auto simp add: is_{sb})
ultimately have load-tmps is_{sb}' $\cap$ dom (j_{sb}(t $\mapsto$ m_{sb} a)) = {}
by auto
from load-tmps-fresh-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' sb' j_{sb}')
qed

from enough-flushs-nth-update [OF i-bound, **where** sb=[]]
have enough-flushs$'$: enough-flushs ts_{sb}'
by (auto simp add: ts_{sb}' sb' sb)

have valid-program-history$'$: valid-program-history ts_{sb}'
proof –
have causal$'$: causal-program-history is_{sb}' sb'
by (simp add: is_{sb} sb sb')
have last-prog p_{sb} sb' = p_{sb}
by (simp add: sb' sb)
from valid-program-history-nth-update [OF i-bound causal$'$ this]
show ?thesis
by (simp add: ts_{sb}' sb')
qed

from is-sim **have** is: is = RMW a t (D,f) cond ret A L R W# is_{sb}'
by (simp add: suspends sb is_{sb})
with ts-i
have ts-i: ts!i = (p_{sb}, RMW a t (D,f) cond ret A L R W# is_{sb}', j_{sb},(),
$\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$, release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
by (simp add: suspends sb)

have flush-all-until-volatile-write ts_{sb} m_{sb} a = m_{sb} a
proof –
have $\forall$ j < length ts_{sb}. i $\neq$ j $\longrightarrow$
(let (-,-,-,sb_j,-,-,-) = ts_{sb}!j
in a $\notin$ outstanding-refs is-non-volatile-$Write_{sb}$ (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$)
sb_j))
proof –
{
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs_j sb_j

assume j-bound: j < length ts_{sb}
assume neq-i-j: i $\neq$ j
assume jth: ts_{sb}!j = (p_j,is_j, xs_j, sb_j, $\mathcal{D}_j$, $\mathcal{O}_j$,$\mathcal{R}_j$)
have a $\notin$ outstanding-refs is-non-volatile-$Write_{sb}$ (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
proof
let ?take-sb_j = (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
let ?drop-sb_j = (dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
assume a-in: a $\in$ outstanding-refs is-non-volatile-$Write_{sb}$?take-sb_j
with outstanding-refs-takeWhile [**where** P$'$= Not $\circ$ is-volatile-$Write_{sb}$]
have a-in$'$: a $\in$ outstanding-refs is-non-volatile-$Write_{sb}$ sb_j
by auto
with non-volatile-owned-or-read-only-outstanding-non-volatile-writes
[OF outstanding-non-volatile-refs-owned-or-read-only [OF j-bound jth]]
have j-owns: a $\in$ $\mathcal{O}_j$ $\cup$ all-acquired sb_j
by auto
from rels-cond [rule-format, OF j-bound [simplified leq] neq-i-j] ts-sim [rule-format, OF j-bound] jth
have no-unsharing:release ?take-sb_j (dom ($\mathcal{S}_{sb}$)) $\mathcal{R}_j$ a $\neq$ Some False
by (auto simp add: Let-def)
from access-cond
show False
proof
assume a $\in$ $\mathcal{O}_{sb}$
with ownership-distinct [OF i-bound j-bound neq-i-j ts_{sb}-i jth]
j-owns
show False
by auto
next
assume a-shared: a $\in$ dom $\mathcal{S}$
with share-all-until-volatile-write-thread-local [OF ownership-distinct-ts_{sb} sharing-consis-ts_{sb} j-bound jth j-owns]
have a-dom: a $\in$ dom (share ?take-sb_j $\mathcal{S}_{sb}$)
by (auto simp add: $\mathcal{S}$ domIff)
from outstanding-non-volatile-writes-unshared [OF j-bound jth]
have non-volatile-writes-unshared $\mathcal{S}_{sb}$ sb_j.
with non-volatile-writes-unshared-append [of $\mathcal{S}_{sb}$ (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
(dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)]
have unshared-take: non-volatile-writes-unshared $\mathcal{S}_{sb}$ (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
by clarsimp

from release-not-unshared-no-write-take [OF unshared-take no-unsharing a-dom] a-in
show False **by** auto
qed
qed
}
thus ?thesis

by (fastforce simp add: Let-def)
qed

from flush-all-until-volatile-write-buffered-val-conv
[OF - i-bound ts_{sb}-i this]
show ?thesis
by (simp add: sb)
qed

hence m-a: m a = m_{sb} a
by (simp add: m)

from cond **have** cond': ¬ cond (j_{sb}(t ↦ m a))
by (simp add: m-a)

from direct-memop-step.RMWReadOnly [**where** cond=cond **and** j=j_{sb} **and** m=m, OF cond']
have (RMW a t (D, f) cond ret A L R W # is_{sb}',
j_{sb}, (),m, $\mathcal{D}$, $\mathcal{O}_{sb}$, $\mathcal{R}_{sb}$, $\mathcal{S}$) →
(is_{sb}', j_{sb}(t ↦ m a), (), m, False, $\mathcal{O}_{sb}$, Map.empty, $\mathcal{S}$).

from direct-computation.concurrent-step.Memop [OF i-bound' ts-i [simplified sb, simplified] this]
have (ts, m, $\mathcal{S}$) $\Rightarrow_d$ (ts[i := (p_{sb}, is_{sb}',
j_{sb}(t ↦ m a), (), False, $\mathcal{O}_{sb}$,Map.empty)], m, $\mathcal{S}$).

moreover

have tmps-commute: j_{sb}(t ↦ (m_{sb} a)) =
(j_{sb} |' (dom j_{sb} − {t}))(t ↦ (m_{sb} a))
apply (rule ext)
apply (auto simp add: restrict-map-def domIff)
done

have (ts_{sb}',m_{sb},$\mathcal{S}_{sb}$') ∼ (ts[i := (p_{sb},is_{sb}', j_{sb}(t ↦ m a),(), False,$\mathcal{O}_{sb}$,Map.empty)],m,$\mathcal{S}$)
apply (rule sim-config.intros)
apply (simp add: ts_{sb}' sb' $\mathcal{O}_{sb}$' $\mathcal{R}_{sb}$' m
flush-all-until-volatile-nth-update-unused [OF i-bound ts_{sb}-i, simplified sb])
using share-all-until-volatile-write-RMW-commute [OF i-bound ts_{sb}-i [simplified is_{sb} sb]]
apply (clarsimp simp add: $\mathcal{S}$ ts_{sb}' $\mathcal{S}_{sb}$' is_{sb} $\mathcal{O}_{sb}$' j_{sb}' sb' sb)
using leq
apply (simp add: ts_{sb}')
using i-bound i-bound' ts-sim
apply (clarsimp simp add: Let-def nth-list-update
ts_{sb}' sb' sb $\mathcal{O}_{sb}$' $\mathcal{R}_{sb}$' $\mathcal{S}_{sb}$' j_{sb}' $\mathcal{D}_{sb}$' ex-not m-a
split: if-split-asm)
apply (rule tmps-commute)
done
ultimately
show ?thesis

using valid-own′ valid-hist′ valid-reads′ valid-sharing′ tmps-distinct′ valid-sops′
valid-dd′ load-tmps-fresh′ enough-flushs′
valid-program-history′ valid′ m_{sb}′ $\mathcal{S}_{sb}$′
by (auto simp del: fun-upd-apply)
next
case (SBHRMWWrite cond t a D f ret A L R W)
then obtain
is_{sb}: is_{sb} = RMW a t (D,f) cond ret A L R W # is_{sb}′ **and**
cond: (cond (j_{sb}(t↦m_{sb} a))) **and**
$\mathcal{O}_{sb}$′: $\mathcal{O}_{sb}$′=$\mathcal{O}_{sb}$ ∪ A − R **and**
$\mathcal{R}_{sb}$′: $\mathcal{R}_{sb}$′=Map.empty **and**
$\mathcal{D}_{sb}$′: ¬ $\mathcal{D}_{sb}$′ **and**
j_{sb}′: j_{sb}′ = j_{sb}(t↦ret (m_{sb} a) (f (j_{sb}(t↦m_{sb} a)))) **and**
sb: sb=[] **and**
sb′: sb′=[] **and**
m_{sb}′: m_{sb}′ = m_{sb}(a := f (j_{sb}(t↦m_{sb} a))) **and**
$\mathcal{S}_{sb}$′: $\mathcal{S}_{sb}$′=$\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L
by auto

from data-dependency-consistent-instrs [OF i-bound ts_{sb}-i]
have D-subset: D ⊆ dom j_{sb}
by (simp add: is_{sb})

from is-sim **have** is: is = RMW a t (D,f) cond ret A L R W # is_{sb}′
by (simp add: suspends sb is_{sb})
with ts-i
have ts-i: ts!i = (p_{sb}, RMW a t (D,f) cond ret A L R W # is_{sb}′, j_{sb},(), $\mathcal{D}$, $\mathcal{O}_{sb}$,$\mathcal{R}_{sb}$)
by (simp add: suspends sb)

from safe-RMW-common [OF safe-memop-flush-sb [simplified is_{sb}]]
obtain access-cond: a ∈ $\mathcal{O}_{sb}$ ∨ a ∈ dom $\mathcal{S}$ **and**
rels-cond: ∀ j < length ts. i≠j ⟶ released (ts!j) a ≠ Some False
by (auto simp add: $\mathcal{S}$ sb)

have a-unflushed:
∀ j < length ts_{sb}. i ≠ j ⟶
(let (-,-,-,sb_j,-,-,-) = ts_{sb}!j
in a ∉ outstanding-refs is-non-volatile-$Write_{sb}$ (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j))
proof −
{
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs_j sb_j
assume j-bound: j < length ts_{sb}
assume neq-i-j: i ≠ j
assume jth: ts_{sb}!j = (p_j,is_j, xs_j, sb_j, $\mathcal{D}_j$, $\mathcal{O}_j$, $\mathcal{R}_j$)
have a ∉ outstanding-refs is-non-volatile-$Write_{sb}$ (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)
proof

let ?take-sb_j = (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)
let ?drop-sb_j = (dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)
assume a-in: a ∈ outstanding-refs is-non-volatile-$Write_{sb}$?take-sb_j
with outstanding-refs-takeWhile [**where** P′= Not ∘ is-volatile-$Write_{sb}$]
have a-in′: a ∈ outstanding-refs is-non-volatile-$Write_{sb}$ sb_j
by auto
with non-volatile-owned-or-read-only-outstanding-non-volatile-writes
[OF outstanding-non-volatile-refs-owned-or-read-only [OF j-bound jth]]
have j-owns: a ∈ $\mathcal{O}_j$ ∪ all-acquired sb_j
by auto
with ownership-distinct [OF i-bound j-bound neq-i-j ts_{sb}-i jth]
have a-not-owns: a ∉ $\mathcal{O}_{sb}$ ∪ all-acquired sb
by blast
assume a-in: a ∈ outstanding-refs is-non-volatile-$Write_{sb}$
(takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)
with outstanding-refs-takeWhile [**where** P′= Not ∘ is-volatile-$Write_{sb}$]
have a-in′: a ∈ outstanding-refs is-non-volatile-$Write_{sb}$ sb_j
by auto
from rels-cond [rule-format, OF j-bound [simplified leq] neq-i-j] ts-sim [rule-format, OF j-bound] jth
have no-unsharing:release ?take-sb_j (dom ($\mathcal{S}_{sb}$)) $\mathcal{R}_j$ a ≠ Some False
by (auto simp add: Let-def)
from access-cond
show False
proof
assume a ∈ $\mathcal{O}_{sb}$
with ownership-distinct [OF i-bound j-bound neq-i-j ts_{sb}-i jth]
j-owns
show False
by auto
next
assume a-shared: a ∈ dom $\mathcal{S}$
with share-all-until-volatile-write-thread-local [OF ownership-distinct-ts_{sb} sharing-consis-ts_{sb} j-bound jth j-owns]
have a-dom: a ∈ dom (share ?take-sb_j $\mathcal{S}_{sb}$)
by (auto simp add: $\mathcal{S}$ domIff)
from outstanding-non-volatile-writes-unshared [OF j-bound jth]
have non-volatile-writes-unshared $\mathcal{S}_{sb}$ sb_j.
with non-volatile-writes-unshared-append [of $\mathcal{S}_{sb}$ (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)
(dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)]
have unshared-take: non-volatile-writes-unshared $\mathcal{S}_{sb}$ (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)
by clarsimp

from release-not-unshared-no-write-take [OF unshared-take no-unsharing a-dom] a-in
show False **by** auto
qed
qed

}
thus ?thesis
by (fastforce simp add: Let-def)
qed

have flush-all-until-volatile-write ts_{sb} m_{sb} a = m_{sb} a
proof –
from flush-all-until-volatile-write-buffered-val-conv
[OF - i-bound ts_{sb}-i a-unflushed]
show ?thesis
by (simp add: sb)
qed

hence m-a: m a = m_{sb} a
by (simp add: m)

from cond **have** cond′: cond (j_{sb}(t $\mapsto$ m a))
by (simp add: m-a)

from safe-memop-flush-sb [simplified is_{sb}] cond′
obtain
L-subset: L $\subseteq$ A **and**
A-shared-owned: A $\subseteq$ dom $\mathcal{S}$ $\cup$ $\mathcal{O}_{sb}$ **and**
R-owned: R $\subseteq$ $\mathcal{O}_{sb}$ **and**
A-R: A $\cap$ R = {} **and**
a-unowned-others-ts:
$\forall$j<length ts. i $\neq$ j $\longrightarrow$ (a $\notin$ owned (ts!j) $\cup$ dom (released (ts!j))) **and**
A-unowned-by-others-ts:
$\forall$j<length ts. i $\neq$ j $\longrightarrow$ (A $\cap$ (owned (ts!j) $\cup$ dom (released (ts!j))) = {}) **and**
a-not-ro: a $\notin$ read-only $\mathcal{S}$
by cases (auto simp add: sb)

from a-unowned-others-ts ts-sim leq
have a-unowned-others:
$\forall$j<length ts_{sb}. i $\neq$ j $\longrightarrow$
(let (-,-,-,sb_j,-,$\mathcal{O}_j$,-) = ts_{sb}!j in
a $\notin$ acquired True (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j) $\mathcal{O}_j$ $\wedge$
a $\notin$ all-shared (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j))
apply (clarsimp simp add: Let-def)
subgoal for j
apply (drule-tac x=j **in** spec)
apply (auto simp add: dom-release-takeWhile)
done
done

from A-unowned-by-others-ts ts-sim leq
have A-unowned-by-others:
$\forall$j<length ts_{sb}. i$\neq$j $\longrightarrow$ (let (-,-,-,sb_j,-,$\mathcal{O}_j$,-) = ts_{sb}!j
in A $\cap$ (acquired True (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j) $\mathcal{O}_j$ $\cup$

all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$)) = {})
apply (clarsimp simp add: Let-def)
subgoal for j
apply (drule-tac x=j **in** spec)
apply (force simp add: dom-release-takeWhile)
done
done

have a-not-ro′: a ∉ read-only $\mathcal{S}_{sb}$
proof
assume a: a ∈ read-only ($\mathcal{S}_{sb}$)
from local.read-only-unowned-axioms **have** read-only-unowned $\mathcal{S}_{sb}$ ts$_{sb}$.
from in-read-only-share-all-until-volatile-write′ [OF ownership-distinct-ts$_{sb}$ sharing-consis-ts$_{sb}$
‹read-only-unowned $\mathcal{S}_{sb}$ ts$_{sb}$› i-bound ts$_{sb}$-i a-unowned-others, simplified sb, simplified,
OF a]
have a ∈ read-only ($\mathcal{S}$)
by (simp add: $\mathcal{S}$)
with a-not-ro **show** False **by** simp
qed

{
fix j
fix p$_j$ is$_{sbj}$ $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_{sbj}$ j$_j$ sb$_j$
assume j-bound: j < length ts$_{sb}$
assume ts$_{sb}$-j: ts$_{sb}$!j=(p$_j$,is$_{sbj}$,j$_j$,sb$_j$,$\mathcal{D}_{sbj}$,$\mathcal{O}_j$,$\mathcal{R}_j$)
assume neq-i-j: i≠j
have a ∉ unforwarded-non-volatile-reads (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$) {}
proof
let ?take-sb$_j$ = takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$
let ?drop-sb$_j$ = dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$
assume a-in: a ∈ unforwarded-non-volatile-reads ?drop-sb$_j$ {}

from a-unowned-others [rule-format, OF - neq-i-j] ts$_{sb}$-j j-bound
obtain a-unacq-take: a ∉ acquired True ?take-sb$_j$ $\mathcal{O}_j$ **and** a-not-shared: a ∉ all-shared ?take-sb$_j$
by auto

note nvo-j = outstanding-non-volatile-refs-owned-or-read-only [OF j-bound ts$_{sb}$-j]

from non-volatile-owned-or-read-only-drop [OF nvo-j]
have nvo-drop-j: non-volatile-owned-or-read-only True (share ?take-sb$_j$ $\mathcal{S}_{sb}$)
(acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?drop-sb$_j$.

note consis-j = sharing-consis [OF j-bound ts$_{sb}$-j]
with sharing-consistent-append [of $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb$_j$?drop-sb$_j$]
obtain consis-take-j: sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb$_j$ **and**
consis-drop-j: sharing-consistent (share ?take-sb$_j$ $\mathcal{S}_{sb}$)

(acquired True ?take-sb_j $\mathcal{O}_j$) ?drop-sb_j
by auto

from in-unforwarded-non-volatile-reads-non-volatile-$Read_{sb}$ [OF a-in]
have a-in′: a ∈ outstanding-refs is-non-volatile-$Read_{sb}$?drop-sb_j.

note reads-consis-j = valid-reads [OF j-bound ts_{sb}-j]
from reads-consistent-drop [OF this]
have reads-consis-drop-j:
reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) (flush ?take-sb_j m_{sb}) ?drop-sb_j.

from read-only-share-all-shared [of a ?take-sb_j $\mathcal{S}_{sb}$] a-not-ro′ a-not-shared
have a-not-ro-j: a ∉ read-only (share ?take-sb_j $\mathcal{S}_{sb}$)
by auto

from ts-sim [rule-format, OF j-bound] ts_{sb}-j j-bound
obtain $suspends_j$ is_j $\mathcal{D}_j$ **where**
$suspends_j$: $suspends_j$ = ?drop-sb_j **and**
is_j: instrs $suspends_j$ @ is_{sbj} = is_j @ prog-instrs $suspends_j$ **and**
$\mathcal{D}_j$: $\mathcal{D}_{sbj}$ = ($\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ sb_j ≠ {}) **and**
ts_j: ts!j = (hd-prog p_j $suspends_j$, is_j,
j_j |‘ (dom j_j − read-tmps $suspends_j$),(),
$\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$,release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
by (auto simp: Let-def)

from ts_j neq-i-j j-bound
have ts′-j: ?ts′!j = (hd-prog p_j $suspends_j$, is_j,
j_j |‘ (dom j_j − read-tmps $suspends_j$),(),
$\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$,release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
by auto

from valid-last-prog [OF j-bound ts_{sb}-j] **have** last-prog: last-prog p_j sb_j = p_j.

from j-bound i-bound′ leq **have** j-bound-ts′: j < length ?ts′
by simp

from read-only-read-acquired-unforwarded-acquire-witness [OF nvo-drop-j consis-drop-j
a-not-ro-j a-unacq-take a-in]
have False
proof
assume ∃sop a′ v ys zs A L R W.
?drop-sb_j= ys @ $Write_{sb}$ True a′ sop v A L R W # zs ∧ a ∈ A ∧
a ∉ outstanding-refs is-$Write_{sb}$ ys ∧ a′≠a
with $suspends_j$
obtain a′ sop′ v′ ys zs′ A′ L′ R′ W′ **where**

split-suspends$_j$: suspends$_j$ = ys @ Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′# zs′ (**is** suspends$_j$=?suspends) **and**
a-A′: a ∈ A′ **and**
no-write: a ∉ outstanding-refs is-Write$_{sb}$ (ys @ [Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′])
by(auto simp add: outstanding-refs-append)

from last-prog
have lp: last-prog p$_j$ suspends$_j$ = p$_j$
apply −
apply (rule last-prog-same-append [**where** sb=?take-sb$_j$])
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
apply simp
done

from sharing-consis [OF j-bound ts$_{sb}$-j]
have sharing-consis-j: sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb$_j$.
then have A′-R′: A′ ∩ R′ = {}
by (simp add: sharing-consistent-append [of - - ?take-sb$_j$?drop-sb$_j$, simplified] suspends$_j$ [symmetric] split-suspends$_j$ sharing-consistent-append)

from valid-program-history [OF j-bound ts$_{sb}$-j]
have causal-program-history is$_{sbj}$ sb$_j$.
then have cph: causal-program-history is$_{sbj}$?suspends
apply −
apply (rule causal-program-history-suffix [**where** sb=?take-sb$_j$])
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
apply (simp add: split-suspends$_j$)
done

from valid-reads [OF j-bound ts$_{sb}$-j]
have reads-consis-j: reads-consistent False $\mathcal{O}_j$ m$_{sb}$ sb$_j$.

from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts$_{sb}$›
j-bound ts$_{sb}$-j this]
have reads-consis-m-j: reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$) m suspends$_j$
by (simp add: m suspends$_j$)

from outstanding-non-write-non-vol-reads-drop-disj [OF i-bound j-bound neq-i-j ts$_{sb}$-i ts$_{sb}$-j]
have outstanding-refs is-Write$_{sb}$?drop-sb ∩ outstanding-refs is-non-volatile-Read$_{sb}$ suspends$_j$ = {}
by (simp add: suspends$_j$)
from reads-consistent-flush-independent [OF this reads-consis-m-j]
have reads-consis-flush-suspend: reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$)
(flush ?drop-sb m) suspends$_j$.

hence reads-consis-ys: reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$)
(flush ?drop-sb m) (ys@[Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′])
by (simp add: split-suspends$_j$ reads-consistent-append)

from valid-write-sops [OF j-bound ts_{sb}-j]
have ∀sop∈write-sops (?take-sb_j@?suspends). valid-sop sop
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain valid-sops-take: ∀sop∈write-sops ?take-sb_j. valid-sop sop **and**
valid-sops-drop: ∀sop∈write-sops (ys@[$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]). valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound ts_{sb}-j]
have distinct-read-tmps (?take-sb_j@$suspends_j$)
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb_j ∩ read-tmps $suspends_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps $suspends_j$
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp only: distinct-read-tmps-append)
done

from valid-history [OF j-bound ts_{sb}-j]
have h-consis:
history-consistent j_j (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof −
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop
h-consis] last-prog-hd-prog
have hist-consis′: history-consistent j_j (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis-j]

have no-vol-read: outstanding-refs is-volatile-Read$_{sb}$
(ys@[Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]) = {}
by (auto simp add: outstanding-refs-append suspends$_j$ [symmetric]
split-suspends$_j$)

have acq-simp:
acquired True (ys @ [Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′])
(acquired True ?take-sb$_j$ $\mathcal{O}_j$) =
acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ∪ A′ − R′
by (simp add: acquired-append)

from flush-store-buffer-append [**where** sb=ys@[Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]
and sb′=zs′, simplified,
OF j-bound-ts′ is$_j$ [simplified split-suspends$_j$] cph [simplified suspends$_j$]
ts′-j [simplified split-suspends$_j$] refl lp [simplified split-suspends$_j$] reads-consis-ys
hist-consis′ [simplified split-suspends$_j$] valid-sops-drop
distinct-read-tmps-drop [simplified split-suspends$_j$]
no-volatile-Read$_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=share ?drop-sb $\mathcal{S}$]

obtain is$_j$′ $\mathcal{R}_j$′ **where**
is$_j$′: instrs zs′ @ is$_{sbj}$ = is$_j$′ @ prog-instrs zs′ **and**
steps-ys: (?ts′, flush ?drop-sb m, share ?drop-sb $\mathcal{S}$) $\Rightarrow_d{}^*$
(?ts′[j:=(last-prog
(hd-prog p$_j$ (Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′# zs′)) (ys@[Write$_{sb}$
True a′ sop′ v′ A′ L′ R′ W′]),
is$_j$′,
j$_j$ |‘ (dom j$_j$ − read-tmps zs′),
(), True, acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ∪ A′ −
R′,$\mathcal{R}_j$′)],
flush (ys@[Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]) (flush ?drop-sb m),
share (ys@[Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]) (share ?drop-sb $\mathcal{S}$))
(**is** (-,-,-) $\Rightarrow_d{}^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto simp add: acquired-append outstanding-refs-append)

from i-bound′ **have** i-bound-ys: i < length ?ts-ys
by auto

from i-bound′ neq-i-j
have ts-ys-i: ?ts-ys!i = (p$_{sb}$, is$_{sb}$, j$_{sb}$,(),
$\mathcal{D}_{sb}$, acquired True sb $\mathcal{O}_{sb}$, release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
by simp
note conflict-computation = rtranclp-trans [OF steps-flush-sb steps-ys]

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe: safe-delayed (?ts-ys,?m-ys,?shared-ys)**.**

from flush-unchanged-addresses [OF no-write]
have flush (ys @ [Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]) m a = m a**.**

with safe-delayedE [OF safe i-bound-ys ts-ys-i, simplified is_{sb}] cond′
have a-unowned:

$\forall$ j < length ?ts-ys. i≠j $\longrightarrow$ (let ($\mathcal{O}_j$) = map owned ?ts-ys!j in a $\notin \mathcal{O}_j$)
apply cases
apply (auto simp add: Let-def is_{sb} sb)
done
from a-A′ a-unowned [rule-format, of j] neq-i-j j-bound leq A′-R′
show False
by (auto simp add: Let-def)
next
assume $\exists$ A L R W ys zs. ?drop-sb_j = ys @ $Ghost_{sb}$ A L R W# zs $\wedge$ a $\in$ A $\wedge$ a $\notin$ outstanding-refs is-$Write_{sb}$ ys
with $suspends_j$
obtain ys zs′ A′ L′ R′ W′ **where**
split-$suspends_j$: $suspends_j$ = ys @ $Ghost_{sb}$ A′ L′ R′ W′# zs′ (**is** $suspends_j$=?suspends) **and**
a-A′: a $\in$ A′ **and**
no-write: a $\notin$ outstanding-refs is-$Write_{sb}$ (ys @ [$Ghost_{sb}$ A′ L′ R′ W′])
by (auto simp add: outstanding-refs-append)

from last-prog
have lp: last-prog p_j $suspends_j$ = p_j
apply –
apply (rule last-prog-same-append [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done
from sharing-consis [OF j-bound ts_{sb}-j]
have sharing-consis-j: sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb_j.
then have A′-R′: A′ $\cap$ R′ = {}
by (simp add: sharing-consistent-append [of - - ?take-sb_j ?drop-sb_j, simplified] $suspends_j$ [symmetric] split-$suspends_j$ sharing-consistent-append)

from valid-program-history [OF j-bound ts_{sb}-j]
have causal-program-history is_{sbj} sb_j.
then have cph: causal-program-history is_{sbj} ?suspends
apply –
apply (rule causal-program-history-suffix [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp add: split-$suspends_j$)
done

from valid-reads [OF j-bound ts_{sb}-j]
have reads-consis-j: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.

from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts_{sb}›
j-bound ts_{sb}-j this]

have reads-consis-m-j: reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$) m suspends$_j$
by (simp add: m suspends$_j$)

from outstanding-non-write-non-vol-reads-drop-disj [OF i-bound j-bound neq-i-j ts$_{sb}$-i ts$_{sb}$-j]
have outstanding-refs is-Write$_{sb}$?drop-sb ∩ outstanding-refs is-non-volatile-Read$_{sb}$ suspends$_j$ = {}
by (simp add: suspends$_j$)
from reads-consistent-flush-independent [OF this reads-consis-m-j]
have reads-consis-flush-suspend: reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$) (flush ?drop-sb m) suspends$_j$.

hence reads-consis-ys: reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$) (flush ?drop-sb m) (ys@[Ghost$_{sb}$ A′ L′ R′ W′])
by (simp add: split-suspends$_j$ reads-consistent-append)

from valid-write-sops [OF j-bound ts$_{sb}$-j]
have ∀sop∈write-sops (?take-sb$_j$@?suspends). valid-sop sop
by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
then obtain valid-sops-take: ∀sop∈write-sops ?take-sb$_j$. valid-sop sop **and**
valid-sops-drop: ∀sop∈write-sops (ys@[Ghost$_{sb}$ A′ L′ R′ W′]). valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound ts$_{sb}$-j]
have distinct-read-tmps (?take-sb$_j$@suspends$_j$)
by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb$_j$ ∩ read-tmps suspends$_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps suspends$_j$
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
apply (simp only: distinct-read-tmps-append)
done

from valid-history [OF j-bound ts$_{sb}$-j]
have h-consis:
history-consistent j$_j$ (hd-prog p$_j$ (?take-sb$_j$@suspends$_j$)) (?take-sb$_j$@suspends$_j$)
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p$_j$ sb$_j$) ?take-sb$_j$ = (hd-prog p$_j$ suspends$_j$)
proof −
from last-prog **have** last-prog p$_j$ (?take-sb$_j$@?drop-sb$_j$) = p$_j$
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p$_j$ suspends$_j$) ?take-sb$_j$ = hd-prog p$_j$ suspends$_j$
by (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
moreover

have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop
h-consis] last-prog-hd-prog
have hist-consis′: history-consistent j_j (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis-j]
have no-vol-read: outstanding-refs is-volatile-$Read_{sb}$
(ys@[$Ghost_{sb}$ A′ L′ R′ W′]) = {}
by (auto simp add: outstanding-refs-append $suspends_j$ [symmetric]
split-$suspends_j$)

have acq-simp:
acquired True (ys @ [$Ghost_{sb}$ A′ L′ R′ W′])
(acquired True ?take-sb_j $\mathcal{O}_j$) =
acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$) ∪ A′ − R′
by (simp add: acquired-append)

from flush-store-buffer-append [**where** sb=ys@[$Ghost_{sb}$ A′ L′ R′ W′] **and** sb′=zs′,
simplified,
OF j-bound-ts′ is_j [simplified split-$suspends_j$] cph [simplified $suspends_j$]
ts′-j [simplified split-$suspends_j$] refl lp [simplified split-$suspends_j$] reads-consis-ys
hist-consis′ [simplified split-$suspends_j$] valid-sops-drop
distinct-read-tmps-drop [simplified split-$suspends_j$]
no-volatile-$Read_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=share ?drop-sb $\mathcal{S}$]

obtain is_j′ $\mathcal{R}_j$′ **where**
is_j′: instrs zs′ @ is_{sbj} = is_j′ @ prog-instrs zs′ **and**
steps-ys: (?ts′, flush ?drop-sb m, share ?drop-sb $\mathcal{S}$) $\Rightarrow_d^*$
(?ts′[j:=(last-prog
(hd-prog p_j ($Ghost_{sb}$ A′ L′ R′ W′# zs′)) (ys@[$Ghost_{sb}$ A′ L′ R′ W′]),
is_j′,
j_j |‘ (dom j_j − read-tmps zs′),
(),
$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ ys ≠ {}, acquired True ys
(acquired True ?take-sb_j $\mathcal{O}_j$) ∪ A′ − R′,$\mathcal{R}_j$′)],
flush (ys@[$Ghost_{sb}$ A′ L′ R′ W′]) (flush ?drop-sb m),
share (ys@[$Ghost_{sb}$ A′ L′ R′ W′]) (share ?drop-sb $\mathcal{S}$))
(**is** (-,-,-) $\Rightarrow_d^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto simp add: acquired-append outstanding-refs-append)

from i-bound′ **have** i-bound-ys: i < length ?ts-ys

by auto

from i-bound′ neq-i-j
have ts-ys-i: ?ts-ys!i = (p_{sb}, is_{sb}, j_{sb},(),
$\mathcal{D}_{sb}$, acquired True sb $\mathcal{O}_{sb}$, release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
by simp
note conflict-computation = rtranclp-trans [OF steps-flush-sb steps-ys]

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe: safe-delayed (?ts-ys,?m-ys,?shared-ys).

from flush-unchanged-addresses [OF no-write]
have flush (ys @ [$Ghost_{sb}$ A′ L′ R′ W′]) m a = m a.

with safe-delayedE [OF safe i-bound-ys ts-ys-i, simplified is_{sb}] cond′
have a-unowned:

$\forall$ j < length ?ts-ys. i≠j ⟶ (let ($\mathcal{O}_j$) = map owned ?ts-ys!j in a $\notin$ $\mathcal{O}_j$)
apply cases
apply (auto simp add: Let-def is_{sb} sb)
done
from a-A′ a-unowned [rule-format, of j] neq-i-j j-bound leq A′-R′

show False
by (auto simp add: Let-def)
qed
then show False
by simp
qed
}
note a-notin-unforwarded-non-volatile-reads-drop = this

have A-unused-by-others:
$\forall$j<length (map $\mathcal{O}$-sb ts_{sb}). i ≠ j ⟶
(let ($\mathcal{O}_j$, sb_j) = map $\mathcal{O}$-sb ts_{sb}! j
in A ∩ ($\mathcal{O}_j$ ∪ outstanding-refs is-volatile-$Write_{sb}$ sb_j) = {})
proof −
{
fix j $\mathcal{O}_j$ sb_j
assume j-bound: j < length (map owned ts_{sb})
assume neq-i-j: i≠j
assume ts_{sb}-j: (map $\mathcal{O}$-sb ts_{sb})!j = ($\mathcal{O}_j$,sb_j)
assume conflict: A ∩ ($\mathcal{O}_j$ ∪ outstanding-refs is-volatile-$Write_{sb}$ sb_j) ≠ {}
have False
proof −
from j-bound leq
have j-bound′: j < length (map owned ts)
by auto
from j-bound **have** j-bound″: j < length ts_{sb}

by auto
from j-bound$'$ **have** j-bound$'''$: j $<$ length ts
by simp

from conflict **obtain** a$'$ **where**
a-in: a$'$ $\in$ A **and**
conflict: a$'$ $\in$ $\mathcal{O}_{j}$ $\vee$ a$'$ $\in$ outstanding-refs is-volatile-Write$_{sb}$ sb$_{j}$
by auto
from A-unowned-by-others [rule-format, OF - neq-i-j] j-bound ts$_{sb}$-j
have A-unshared-j: A $\cap$ all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_{j}$) =
{}
by (auto simp add: Let-def)
from conflict
show ?thesis
proof

assume a$'$ $\in$ $\mathcal{O}_{j}$

from all-shared-acquired-in [OF this] A-unshared-j a-in
have conflict: a$'$ $\in$ acquired True (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_{j}$) $\mathcal{O}_{j}$
by (auto)
with A-unowned-by-others [rule-format, OF - neq-i-j] j-bound ts$_{sb}$-j a-in
show False **by** auto
next
assume a-in-j: a$'$ $\in$ outstanding-refs is-volatile-Write$_{sb}$ sb$_{j}$

let ?take-sb$_{j}$ = (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_{j}$)
let ?drop-sb$_{j}$ = (dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_{j}$)

from ts-sim [rule-format, OF j-bound$''$] ts$_{sb}$-j j-bound$''$
obtain p$_{j}$ suspends$_{j}$ is$_{sbj}$ $\mathcal{D}_{sbj}$ $\mathcal{D}_{j}$ $\mathcal{R}_{j}$ j$_{sbj}$ is$_{j}$ **where**
ts$_{sb}$-j: ts$_{sb}$! j = (p$_{j}$,is$_{sbj}$, j$_{sbj}$, sb$_{j}$,$\mathcal{D}_{sbj}$,$\mathcal{O}_{j}$,$\mathcal{R}_{j}$) **and**
suspends$_{j}$: suspends$_{j}$ = ?drop-sb$_{j}$ **and**
$\mathcal{D}_{j}$: $\mathcal{D}_{sbj}$ = ($\mathcal{D}_{j}$ $\vee$ outstanding-refs is-volatile-Write$_{sb}$ sb$_{j}$ $\neq$ {}) **and**
is$_{j}$: instrs suspends$_{j}$ @ is$_{sbj}$ = is$_{j}$ @ prog-instrs suspends$_{j}$ **and**
ts$_{j}$: ts!j = (hd-prog p$_{j}$ suspends$_{j}$, is$_{j}$,
j$_{sbj}$ |‘ (dom j$_{sbj}$ $-$ read-tmps suspends$_{j}$),(), $\mathcal{D}_{j}$, acquired True ?take-sb$_{j}$ $\mathcal{O}_{j}$,release
?take-sb$_{j}$ (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{j}$)
apply (cases ts$_{sb}$!j)
apply (force simp add: Let-def)
done

have a$'$ $\in$ outstanding-refs is-volatile-Write$_{sb}$ suspends$_{j}$
proof $-$
from a-in-j
have a$'$ $\in$ outstanding-refs is-volatile-Write$_{sb}$ (?take-sb$_{j}$ @ ?drop-sb$_{j}$)
by simp
thus ?thesis

apply (simp only: outstanding-refs-append $suspends_j$)
apply (auto simp add: outstanding-refs-conv dest: set-takeWhileD)
done
qed

from split-volatile-$Write_{sb}$-in-outstanding-refs [OF this]
obtain sop′ v′ ys zs A′ L′ R′ W′ **where**
split-$suspends_j$: $suspends_j$ = ys @ $Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′# zs (**is** $suspends_j$ = ?suspends)
by blast

from valid-program-history [OF j-bound″ ts_{sb}-j]
have causal-program-history is_{sbj} sb_j.
then have cph: causal-program-history is_{sbj} ?suspends
apply –
apply (rule causal-program-history-suffix [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp add: split-$suspends_j$)
done

from valid-last-prog [OF j-bound″ ts_{sb}-j] **have** last-prog: last-prog p_j sb_j = p_j.
then
have lp: last-prog p_j ?suspends = p_j
apply –
apply (rule last-prog-same-append [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

from valid-reads [OF j-bound″ ts_{sb}-j]
have reads-consis: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.
from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts_{sb}›
j-bound″
ts_{sb}-j this]
have reads-consis-m-j: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
by (simp add: m $suspends_j$)

from outstanding-non-volatile-refs-owned-or-read-only [OF j-bound″ ts_{sb}-j]
have nvo-j: non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb_j.
with non-volatile-owned-or-read-only-append [of False $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb_j ?drop-sb_j]
have nvo-take-j: non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb_j
by auto

from a-unowned-others [rule-format, OF - neq-i-j] ts_{sb}-j j-bound
have a-not-acq: a $\notin$ acquired True ?take-sb_j $\mathcal{O}_j$
by auto

from a-notin-unforwarded-non-volatile-reads-drop[OF j-bound′′ ts_{sb}-j neq-i-j]
have a-notin-unforwarded-reads: a $\notin$ unforwarded-non-volatile-reads $suspends_j$ {}
by (simp add: $suspends_j$)

let ?ma=(m(a := f (j_{sb}(t$\mapsto$m a))))

from reads-consistent-mem-eq-on-unforwarded-non-volatile-reads [**where** W={}
and m′=?ma,simplified, OF - subset-refl reads-consis-m-j]
a-notin-unforwarded-reads
have reads-consis-ma-j:
reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) ?ma $suspends_j$
by auto

from reads-consis-ma-j
have reads-consis-ys: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) ?ma (ys)
by (simp add: split-$suspends_j$ reads-consistent-append)

from direct-memop-step.RMWWrite [**where** cond=cond **and** j=j_{sb} **and** m=m, OF cond′]
have (RMW a t (D, f) cond ret A L R W# is_{sb}′, j_{sb}, (), m,$\mathcal{D}$, $\mathcal{O}_{sb}$, $\mathcal{R}_{sb}$, $\mathcal{S}$) $\rightarrow$
(is_{sb}′, j_{sb}(t $\mapsto$ ret (m a) (f (j_{sb}(t $\mapsto$ m a)))), (), ?ma, False, $\mathcal{O}_{sb}$ $\cup$ A $-$ R, Map.empty,$\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L).
from direct-computation.concurrent-step.Memop [OF i-bound′ ts-i this]
have step-a: (ts, m, $\mathcal{S}$) $\Rightarrow_d$
(ts[i := (p_{sb}, is_{sb}′, j_{sb}(t $\mapsto$ ret (m a) (f (j_{sb}(t $\mapsto$ m a)))), (), False, $\mathcal{O}_{sb}$ $\cup$ A $-$ R,Map.empty)],
?ma,$\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)
(**is** - $\Rightarrow_d$ (?ts-a, -, ?shared-a)).

from ts_j neq-i-j j-bound

have ts-a-j: ?ts-a!j = (hd-prog p_j $suspends_j$, is_j,
j_{sbj} |‘ (dom j_{sbj} $-$ read-tmps $suspends_j$),(), $\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$,release ?take-sb_j (dom ($\mathcal{S}_{sb}$)) $\mathcal{R}_j$)
by auto

from valid-write-sops [OF j-bound′′ ts_{sb}-j]
have $\forall$sop$\in$write-sops (?take-sb_j@?suspends). valid-sop sop
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain valid-sops-take: $\forall$sop$\in$write-sops ?take-sb_j. valid-sop sop **and**
valid-sops-drop: $\forall$sop$\in$write-sops (ys). valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound′′ ts_{sb}-j]
have distinct-read-tmps (?take-sb_j@$suspends_j$)
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain

read-tmps-take-drop: read-tmps ?take-sb_j ∩ read-tmps $suspends_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps $suspends_j$
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp only: distinct-read-tmps-append)
done

from valid-history [OF j-bound′′ ts_{sb}-j]
have h-consis:
history-consistent j_{sbj} (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof −
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop
h-consis] last-prog-hd-prog
have hist-consis′: history-consistent j_{sbj} (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis]
have no-vol-read: outstanding-refs is-volatile-$Read_{sb}$ (ys) = {}
by (auto simp add: outstanding-refs-append $suspends_j$ [symmetric]
split-$suspends_j$)
from j-bound′ **have** j-bound-ts-a: j < length ?ts-a **by** auto

from flush-store-buffer-append [**where** sb=ys **and** sb′=$Write_{sb}$ True a′ sop′ v′ A′ L′
R′ W′#zs, simplified,
OF j-bound-ts-a is_j [simplified split-$suspends_j$] cph [simplified $suspends_j$]
ts-a-j [simplified split-$suspends_j$] refl lp [simplified split-$suspends_j$] reads-consis-ys
hist-consis′ [simplified split-$suspends_j$] valid-sops-drop
distinct-read-tmps-drop [simplified split-$suspends_j$]
no-volatile-$Read_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=?shared-a]

obtain $is_j{}'$ $\mathcal{R}_j{}'$ **where**
$is_j{}'$: Write True a′ sop′ A′ L′ R′ W′# instrs zs @ is_{sbj} = $is_j{}'$ @ prog-instrs zs **and**
steps-ys: (?ts-a, ?ma, ?shared-a) $\Rightarrow_d{}^*$
(?ts-a[j:=(last-prog
(hd-prog p_j zs) ys,
$is_j{}'$,
j_{sbj} |‘ (dom j_{sbj} − read-tmps zs),
(), $\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ ys ≠ {}, acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$),$\mathcal{R}_j{}'$)],
flush ys (?ma), share ys (?shared-a))
(**is** (-,-,-) $\Rightarrow_d{}^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto simp add: acquired-append)

from cph
have causal-program-history is_{sbj} ((ys @ [$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]) @ zs)
by simp
from causal-program-history-suffix [OF this]
have cph′: causal-program-history is_{sbj} zs.
interpret $causal_j$: causal-program-history is_{sbj} zs **by** (rule cph′)

from $causal_j$.causal-program-history [of [], simplified, OF refl] $is_j{}'$
obtain $is_j{}''$
where $is_j{}'$: $is_j{}'$ = Write True a′ sop′ A′ L′ R′ W′#$is_j{}''$ **and**
$is_j{}''$: instrs zs @ is_{sbj} = $is_j{}''$ @ prog-instrs zs
by clarsimp

from i-bound′ **have** i-bound-ys: i < length ?ts-ys
by auto

from i-bound′ neq-i-j
have ts-ys-i: ?ts-ys!i = (p_{sb}, $is_{sb}{}'$,
j_{sb}(t ↦ ret (m a) (f (j_{sb}(t ↦ m a)))),(), False, $\mathcal{O}_{sb}$ ∪ A − R,Map.empty)
by simp

from j-bound-ts-a **have** j-bound-ys: j < length ?ts-ys
by auto
then have ts-ys-j: ?ts-ys!j = (last-prog (hd-prog p_j zs) ys, Write True a′ sop′ A′ L′ R′ W′#$is_j{}''$, j_{sbj} |‘ (dom j_{sbj} − read-tmps zs), (), $\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ ys ≠ {},
acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$),$\mathcal{R}_j{}'$)
by (clarsimp simp add: $is_j{}'$)
note conflict-computation = r-rtranclp-rtranclp [OF step-a steps-ys]

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys).

from safe-delayedE [OF this j-bound-ys ts-ys-j]
have a-unowned:
∀i < length ts. j≠i ⟶ (let ($\mathcal{O}_i$) = map owned ?ts-ys!i in a′ ∉ $\mathcal{O}_i$)

apply cases
apply (auto simp add: Let-def)
done
from a-in a-unowned [rule-format, of i] neq-i-j i-bound′ A-R
show False
by (auto simp add: Let-def)
qed
qed
}
thus ?thesis
by (auto simp add: Let-def)
qed

have A-unacquired-by-others:
$\forall$ j<length (map $\mathcal{O}$-sb ts_{sb}). i $\neq$ j $\longrightarrow$
(let ($\mathcal{O}_j$, sb_j) = map $\mathcal{O}$-sb ts_{sb}! j
in A $\cap$ all-acquired sb_j = {})
proof −
{
fix j $\mathcal{O}_j$ sb_j
assume j-bound: j < length (map owned ts_{sb})
assume neq-i-j: i$\neq$j
assume ts_{sb}-j: (map $\mathcal{O}$-sb ts_{sb})!j = ($\mathcal{O}_j$,sb_j)
assume conflict: A $\cap$ all-acquired sb_j $\neq$ {}
have False
proof −
from j-bound leq
have j-bound′: j < length (map owned ts)
by auto
from j-bound **have** j-bound″: j < length ts_{sb}
by auto
from j-bound′ **have** j-bound‴: j < length ts
by simp

from conflict **obtain** a′ **where**
a′-in: a′ $\in$ A **and**
a′-in-j: a′ $\in$ all-acquired sb_j
by auto

let ?take-sb_j = (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
let ?drop-sb_j = (dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)

from ts-sim [rule-format, OF j-bound″] ts_{sb}-j j-bound″
obtain p_j $suspends_j$ is_{sbj} j_{sbj} $\mathcal{D}_{sbj}$ $\mathcal{R}_j$ $\mathcal{D}_j$ is_j **where**
ts_{sb}-j: ts_{sb} ! j = (p_j,is_{sbj},j_{sbj}, sb_j,$\mathcal{D}_{sbj}$,$\mathcal{O}_j$,$\mathcal{R}_j$) **and**
$suspends_j$: $suspends_j$ = ?drop-sb_j **and**
is_j: instrs $suspends_j$ @ is_{sbj} = is_j @ prog-instrs $suspends_j$ **and**
$\mathcal{D}_j$: $\mathcal{D}_{sbj}$ = ($\mathcal{D}_j$ $\vee$ outstanding-refs is-volatile-$Write_{sb}$ sb_j $\neq$ {}) **and**
ts_j: ts!j = (hd-prog p_j $suspends_j$, is_j,
j_{sbj} |‘ (dom j_{sbj} − read-tmps $suspends_j$),(),

$\mathcal{D}_j$, acquired True ?take-sb$_j$ $\mathcal{O}_j$,release ?take-sb$_j$ (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
apply (cases ts$_{sb}$!j)
apply (force simp add: Let-def)
done

from a′-in-j all-acquired-append [of ?take-sb$_j$?drop-sb$_j$]
have a′ ∈ all-acquired ?take-sb$_j$ ∨ a′ ∈ all-acquired suspends$_j$
by (auto simp add: suspends$_j$)
thus False
proof
assume a′ ∈ all-acquired ?take-sb$_j$
with A-unowned-by-others [rule-format, OF - neq-i-j] ts$_{sb}$-j j-bound a′-in
show False
by (auto dest: all-acquired-unshared-acquired)
next
assume conflict-drop: a′ ∈ all-acquired suspends$_j$

from split-all-acquired-in [OF conflict-drop]
show ?thesis
proof
assume ∃sop a″ v ys zs A L R W.
suspends$_j$ = ys @ Write$_{sb}$ True a″ sop v A L R W# zs ∧ a′ ∈ A
then
obtain a″ sop′ v′ ys zs A′ L′ R′ W′ **where**
split-suspends$_j$: suspends$_j$ = ys @ Write$_{sb}$ True a″ sop′ v′ A′ L′ R′ W′# zs (**is** suspends$_j$ = ?suspends) **and**
a′-A′: a′ ∈ A′
by blast

from valid-program-history [OF j-bound″ ts$_{sb}$-j]
have causal-program-history is$_{sbj}$ sb$_j$.
then have cph: causal-program-history is$_{sbj}$?suspends
apply –
apply (rule causal-program-history-suffix [**where** sb=?take-sb$_j$])
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
apply (simp add: split-suspends$_j$)
done

from valid-last-prog [OF j-bound″ ts$_{sb}$-j] **have** last-prog: last-prog p$_j$ sb$_j$ = p$_j$.
then
have lp: last-prog p$_j$?suspends = p$_j$
apply –
apply (rule last-prog-same-append [**where** sb=?take-sb$_j$])
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
apply simp
done

from valid-reads [OF j-bound″ ts$_{sb}$-j]

have reads-consis: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.
from reads-consistent-flush-all-until-volatile-write [OF
‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts_{sb}› j-bound″
ts_{sb}-j this]
have reads-consis-m-j:
reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
by (simp add: m $suspends_j$)

from outstanding-non-volatile-refs-owned-or-read-only [OF j-bound″ ts_{sb}-j]
have nvo-j: non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb_j.
with non-volatile-owned-or-read-only-append [of False $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb_j ?drop-sb_j]
have nvo-take-j: non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb_j
by auto

from a-unowned-others [rule-format, OF - neq-i-j] ts_{sb}-j j-bound
have a-not-acq: a $\notin$ acquired True ?take-sb_j $\mathcal{O}_j$
by auto

from a-notin-unforwarded-non-volatile-reads-drop[OF j-bound″ ts_{sb}-j neq-i-j]
have a-notin-unforwarded-reads: a $\notin$ unforwarded-non-volatile-reads $suspends_j$ {}
by (simp add: $suspends_j$)

let ?ma=(m(a := f (j_{sb}(t↦m a))))

from reads-consistent-mem-eq-on-unforwarded-non-volatile-reads [**where** W={}
and m′=?ma,simplified, OF - subset-refl reads-consis-m-j]
a-notin-unforwarded-reads
have reads-consis-ma-j:
reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) ?ma $suspends_j$
by auto

from reads-consis-ma-j
have reads-consis-ys: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) ?ma (ys)
by (simp add: split-$suspends_j$ reads-consistent-append)

from direct-memop-step.RMWWrite [**where** cond=cond **and** j=j_{sb} **and** m=m, OF cond′]
have (RMW a t (D, f) cond ret A L R W# is_{sb}′,
j_{sb}, (), m, $\mathcal{D}$, $\mathcal{O}_{sb}$, $\mathcal{R}_{sb}$, $\mathcal{S}$) →
(is_{sb}′,
j_{sb}(t ↦ ret (m a) (f (j_{sb}(t ↦ m a)))), (), ?ma, False, $\mathcal{O}_{sb}$ ∪ A − R,Map.empty, $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L).
from direct-computation.concurrent-step.Memop [OF i-bound′ ts-i [simplified sb, simplified] this]
have step-a: (ts, m, $\mathcal{S}$) $\Rightarrow_d$
(ts[i := (p_{sb}, is_{sb}′, j_{sb}(t ↦ ret (m a) (f (j_{sb}(t ↦ m a)))), (), False, $\mathcal{O}_{sb}$ ∪ A − R,Map.empty)],
?ma,$\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)

(**is** - $\Rightarrow_{\mathsf{d}}$ (?ts-a, -, ?shared-a)).

from $\mathrm{ts_j}$ neq-i-j j-bound

have ts-a-j: ?ts-a!j = (hd-prog $\mathrm{p_j}$ $\mathrm{suspends_j}$, $\mathrm{is_j}$,
$\mathsf{j_{sbj}}$ |' (dom $\mathsf{j_{sbj}}$ − read-tmps $\mathrm{suspends_j}$),(),
$\mathcal{D}_\mathrm{j}$, acquired True ?take-$\mathrm{sb_j}$ $\mathcal{O}_\mathrm{j}$,release ?take-$\mathrm{sb_j}$ (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}_\mathrm{j}$)
by auto

from valid-write-sops [OF j-bound$''$ $\mathrm{ts_{sb}}$-j]
have ∀sop∈write-sops (?take-$\mathrm{sb_j}$@?suspends). valid-sop sop
by (simp add: split-$\mathrm{suspends_j}$ [symmetric] $\mathrm{suspends_j}$)
then obtain valid-sops-take: ∀sop∈write-sops ?take-$\mathrm{sb_j}$. valid-sop sop **and**
valid-sops-drop: ∀sop∈write-sops (ys). valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound$''$ $\mathrm{ts_{sb}}$-j]
have distinct-read-tmps (?take-$\mathrm{sb_j}$@$\mathrm{suspends_j}$)
by (simp add: split-$\mathrm{suspends_j}$ [symmetric] $\mathrm{suspends_j}$)
then obtain
read-tmps-take-drop: read-tmps ?take-$\mathrm{sb_j}$ ∩ read-tmps $\mathrm{suspends_j}$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps $\mathrm{suspends_j}$
apply (simp only: split-$\mathrm{suspends_j}$ [symmetric] $\mathrm{suspends_j}$)
apply (simp only: distinct-read-tmps-append)
done

from valid-history [OF j-bound$''$ $\mathrm{ts_{sb}}$-j]
have h-consis:
history-consistent $\mathsf{j_{sbj}}$ (hd-prog $\mathrm{p_j}$ (?take-$\mathrm{sb_j}$@$\mathrm{suspends_j}$)) (?take-$\mathrm{sb_j}$@$\mathrm{suspends_j}$)
apply (simp only: split-$\mathrm{suspends_j}$ [symmetric] $\mathrm{suspends_j}$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog $\mathrm{p_j}$ $\mathrm{sb_j}$) ?take-$\mathrm{sb_j}$ = (hd-prog $\mathrm{p_j}$ $\mathrm{suspends_j}$)
proof −
from last-prog **have** last-prog $\mathrm{p_j}$ (?take-$\mathrm{sb_j}$@?drop-$\mathrm{sb_j}$) = $\mathrm{p_j}$
by simp
from last-prog-hd-prog-append$'$ [OF h-consis] this
have last-prog (hd-prog $\mathrm{p_j}$ $\mathrm{suspends_j}$) ?take-$\mathrm{sb_j}$ = hd-prog $\mathrm{p_j}$ $\mathrm{suspends_j}$
by (simp only: split-$\mathrm{suspends_j}$ [symmetric] $\mathrm{suspends_j}$)
moreover
have last-prog (hd-prog $\mathrm{p_j}$ (?take-$\mathrm{sb_j}$ @ $\mathrm{suspends_j}$)) ?take-$\mathrm{sb_j}$ =
last-prog (hd-prog $\mathrm{p_j}$ $\mathrm{suspends_j}$) ?take-$\mathrm{sb_j}$
apply (simp only: split-$\mathrm{suspends_j}$ [symmetric] $\mathrm{suspends_j}$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis

by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
qed

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop h-consis] last-prog-hd-prog
have hist-consis$'$: history-consistent j_{sbj} (hd-prog p_j suspends$_j$) suspends$_j$
by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis]
have no-vol-read: outstanding-refs is-volatile-Read$_{sb}$ (ys) = {}
by (auto simp add: outstanding-refs-append suspends$_j$ [symmetric] split-suspends$_j$)
from j-bound$'$ **have** j-bound-ts-a: j < length ?ts-a **by** auto

from flush-store-buffer-append [**where** sb=ys **and** sb$'$=Write$_{sb}$ True a$''$ sop$'$ v$'$ A$'$ L$'$ R$'$ W$'$#zs, simplified,
OF j-bound-ts-a is$_j$ [simplified split-suspends$_j$] cph [simplified suspends$_j$]
ts-a-j [simplified split-suspends$_j$] refl lp [simplified split-suspends$_j$] reads-consis-ys
hist-consis$'$ [simplified split-suspends$_j$] valid-sops-drop
distinct-read-tmps-drop [simplified split-suspends$_j$]
no-volatile-Read$_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=?shared-a]

obtain is$_j{}'$ $\mathcal{R}_j{}'$ **where**
is$_j{}'$: Write True a$''$ sop$'$ A$'$ L$'$ R$'$ W$'$# instrs zs @ is$_{sbj}$ = is$_j{}'$ @ prog-instrs zs **and**
steps-ys: (?ts-a, ?ma, ?shared-a) $\Rightarrow_d{}^*$
(?ts-a[j:=(last-prog
(hd-prog p_j zs) ys,
is$_j{}'$,
j_{sbj} |‘ (dom j_{sbj} − read-tmps zs),
(),
$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-Write$_{sb}$ ys ≠ {}, acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$),$\mathcal{R}_j{}'$)],
flush ys (?ma),
share ys (?shared-a))
(**is** (-,-,-) $\Rightarrow_d{}^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto simp add: acquired-append)

from cph
have causal-program-history is$_{sbj}$ ((ys @ [Write$_{sb}$ True a$''$ sop$'$ v$'$ A$'$ L$'$ R$'$ W$'$]) @ zs)
by simp
from causal-program-history-suffix [OF this]
have cph$'$: causal-program-history is$_{sbj}$ zs.
interpret causal$_j$: causal-program-history is$_{sbj}$ zs **by** (rule cph$'$)

from causal$_j$.causal-program-history [of [], simplified, OF refl] is$_j{}'$
obtain is$_j{}''$
where is$_j{}'$: is$_j{}'$ = Write True a$''$ sop$'$ A$'$ L$'$ R$'$ W$'$#is$_j{}''$ **and**
is$_j{}''$: instrs zs @ is$_{sbj}$ = is$_j{}''$ @ prog-instrs zs
by clarsimp

from i-bound′ **have** i-bound-ys: i < length ?ts-ys
  by auto

from i-bound′ neq-i-j
have ts-ys-i: ?ts-ys!i = (p_{sb}, is_{sb}′,
  j_{sb}(t ↦ ret (m a) (f (j_{sb}(t ↦ m a)))),(), False, $\mathcal{O}_{sb}$ ∪ A − R,Map.empty)
  by simp

from j-bound-ts-a **have** j-bound-ys: j < length ?ts-ys
  by auto
then have ts-ys-j: ?ts-ys!j = (last-prog (hd-prog p_j zs) ys, Write True a″ sop′ A′ L′ R′ W′#is_j″,
  j_{sbj} |‘ (dom j_{sbj} − read-tmps zs), (),
  $\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ ys ≠ {},
  acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$),$\mathcal{R}_j$′)
  by (clarsimp simp add: is_j′)
note conflict-computation = r-rtranclp-rtranclp [OF step-a steps-ys]

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys)**.**

from safe-delayedE [OF this j-bound-ys ts-ys-j]
have A′-unowned:
  ∀i < length ?ts-ys. j≠i ⟶ (let ($\mathcal{O}_i$) = map owned ?ts-ys!i in A′ ∩ $\mathcal{O}_i$ = {})
  apply cases
  apply (fastforce simp add: Let-def is_{sb})+
  done
from a′-in a′-A′ A′-unowned [rule-format, of i] neq-i-j i-bound′ A-R
show False
  by (auto simp add: Let-def)
    next
assume ∃A L R W ys zs. $suspends_j$ = ys @ $Ghost_{sb}$ A L R W# zs ∧ a′ ∈ A
then
obtain ys zs A′ L′ R′ W′ **where**
  split-$suspends_j$: $suspends_j$ = ys @ $Ghost_{sb}$ A′ L′ R′ W′# zs (**is** $suspends_j$ = ?suspends)
and
  a′-A′: a′ ∈ A′
  by blast

from valid-program-history [OF j-bound″ ts_{sb}-j]
have causal-program-history is_{sbj} sb_j**.**
then have cph: causal-program-history is_{sbj} ?suspends
  apply −
  apply (rule causal-program-history-suffix [**where** sb=?take-sb_j])
  apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
  apply (simp add: split-$suspends_j$)
  done

from valid-last-prog [OF j-bound'' ts_{sb}-j] **have** last-prog: last-prog p_j sb_j = p_j.
then
have lp: last-prog p_j ?suspends = p_j
  apply –
  apply (rule last-prog-same-append [**where** sb=?take-sb_j])
  apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
  apply simp
  done

from valid-reads [OF j-bound'' ts_{sb}-j]
have reads-consis: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.
from reads-consistent-flush-all-until-volatile-write [OF
  ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts_{sb}› j-bound''
  ts_{sb}-j this]
have reads-consis-m-j:
  reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
  by (simp add: m $suspends_j$)

from outstanding-non-volatile-refs-owned-or-read-only [OF j-bound'' ts_{sb}-j]
have nvo-j: non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb_j.
with non-volatile-owned-or-read-only-append [of False $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb_j ?drop-sb_j]
have nvo-take-j: non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb_j
  by auto

from a-unowned-others [rule-format, OF - neq-i-j] ts_{sb}-j j-bound
have a-not-acq: a $\notin$ acquired True ?take-sb_j $\mathcal{O}_j$
  by auto

from a-notin-unforwarded-non-volatile-reads-drop[OF j-bound'' ts_{sb}-j neq-i-j]
have a-notin-unforwarded-reads: a $\notin$ unforwarded-non-volatile-reads $suspends_j$ {}
  by (simp add: $suspends_j$)

let ?ma=(m(a := f (j_{sb}(t↦m a))))

from reads-consistent-mem-eq-on-unforwarded-non-volatile-reads [**where** W={}
  and m'=?ma,simplified, OF - subset-refl reads-consis-m-j]
  a-notin-unforwarded-reads
have reads-consis-ma-j:
  reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) ?ma $suspends_j$
  by auto

from reads-consis-ma-j
have reads-consis-ys: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) ?ma (ys)
  by (simp add: split-$suspends_j$ reads-consistent-append)

from direct-memop-step.RMWWrite [**where** cond=cond **and** j=j_{sb} **and** m=m, OF cond']

have (RMW a t (D, f) cond ret A L R W# is_{sb}',
j_{sb}, (), m, $\mathcal{D}$,$\mathcal{O}_{sb}$, $\mathcal{R}_{sb}$, $\mathcal{S}$) $\rightarrow$
(is_{sb}',
j_{sb}(t $\mapsto$ ret (m a) (f (j_{sb}(t $\mapsto$ m a)))), (), ?ma, False, $\mathcal{O}_{sb}$ $\cup$ A $-$ R,Map.empty,$\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L).
from direct-computation.concurrent-step.Memop [OF i-bound′ ts-i [simplified sb, simplified] this]
have step-a: (ts, m, $\mathcal{S}$) $\Rightarrow_d$
(ts[i := (p_{sb}, is_{sb}', j_{sb}(t $\mapsto$ ret (m a) (f (j_{sb}(t $\mapsto$ m a)))), (), False, $\mathcal{O}_{sb}$ $\cup$ A $-$ R,Map.empty)],
?ma,$\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)
(**is** - $\Rightarrow_d$ (?ts-a, -, ?shared-a)).

from ts_j neq-i-j j-bound

have ts-a-j: ?ts-a!j = (hd-prog p_j $suspends_j$, is_j,
j_{sbj} |‘ (dom j_{sbj} $-$ read-tmps $suspends_j$),(), $\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$,release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
by auto

from valid-write-sops [OF j-bound″ ts_{sb}-j]
have $\forall$sop$\in$write-sops (?take-sb_j@?suspends). valid-sop sop
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain valid-sops-take: $\forall$sop$\in$write-sops ?take-sb_j. valid-sop sop **and**
valid-sops-drop: $\forall$sop$\in$write-sops (ys). valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound″ ts_{sb}-j]
have distinct-read-tmps (?take-sb_j@$suspends_j$)
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb_j $\cap$ read-tmps $suspends_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps $suspends_j$
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp only: distinct-read-tmps-append)
done

from valid-history [OF j-bound″ ts_{sb}-j]
have h-consis:
history-consistent j_{sbj} (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof $-$

from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop
h-consis] last-prog-hd-prog
have hist-consis′: history-consistent j_{sbj} (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis]
have no-vol-read: outstanding-refs is-volatile-$Read_{sb}$ (ys) = {}
by (auto simp add: outstanding-refs-append $suspends_j$ [symmetric]
split-$suspends_j$)
from j-bound′ **have** j-bound-ts-a: j < length ?ts-a **by** auto

from flush-store-buffer-append [**where** sb=ys **and** sb′=$Ghost_{sb}$ A′ L′ R′ W′#zs, simplified,
OF j-bound-ts-a is_j [simplified split-$suspends_j$] cph [simplified $suspends_j$]
ts-a-j [simplified split-$suspends_j$] refl lp [simplified split-$suspends_j$] reads-consis-ys
hist-consis′ [simplified split-$suspends_j$] valid-sops-drop
distinct-read-tmps-drop [simplified split-$suspends_j$]
no-volatile-$Read_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=?shared-a]

obtain is_j′ $\mathcal{R}_j$′ **where**
is_j′: Ghost A′ L′ R′ W′# instrs zs @ is_{sbj} = is_j′ @ prog-instrs zs **and**
steps-ys: (?ts-a, ?ma, ?shared-a) $\Rightarrow_d{}^*$
(?ts-a[j:=(last-prog
(hd-prog p_j zs) ys,
is_j′,
j_{sbj} |‘ (dom j_{sbj} − read-tmps zs),
(),
$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ ys ≠ {}, acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$),$\mathcal{R}_j$′)],
flush ys (?ma),
share ys (?shared-a))
(**is** (-,-,-) $\Rightarrow_d{}^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto simp add: acquired-append)

from cph

have causal-program-history is_{sbj} ((ys @ [$Ghost_{sb}$ A′ L′ R′ W′]) @ zs)
by simp
from causal-program-history-suffix [OF this]
have cph′: causal-program-history is_{sbj} zs.
interpret $causal_j$: causal-program-history is_{sbj} zs **by** (rule cph′)

from $causal_j$.causal-program-history [of [], simplified, OF refl] is_j′
obtain is_j″
where is_j′: is_j′ = Ghost A′ L′ R′ W′#is_j″ **and**
is_j″: instrs zs @ is_{sbj} = is_j″ @ prog-instrs zs
by clarsimp

from i-bound′ **have** i-bound-ys: i < length ?ts-ys
by auto

from i-bound′ neq-i-j
have ts-ys-i: ?ts-ys!i = (p_{sb}, is_{sb}′,
j_{sb}(t ↦ ret (m a) (f (j_{sb}(t ↦ m a)))),(), False, $\mathcal{O}_{sb}$ ∪ A − R,Map.empty)
by simp

from j-bound-ts-a **have** j-bound-ys: j < length ?ts-ys
by auto
then have ts-ys-j: ?ts-ys!j = (last-prog (hd-prog p_j zs) ys, Ghost A′ L′ R′ W′#is_j″,
j_{sbj} |‘ (dom j_{sbj} − read-tmps zs), (),
$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ ys ≠ {},
acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$),$\mathcal{R}_j$′)
by (clarsimp simp add: is_j′)
note conflict-computation = r-rtranclp-rtranclp [OF step-a steps-ys]

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys).

from safe-delayedE [OF this j-bound-ys ts-ys-j]
have A′-unowned:
∀ i < length ?ts-ys. j≠i ⟶ (let ($\mathcal{O}_i$) = map owned ?ts-ys!i in A′ ∩ $\mathcal{O}_i$ = {})
apply cases
apply (fastforce simp add: Let-def is_{sb})+
done
from a′-in a′-A′ A′-unowned [rule-format, of i] neq-i-j i-bound′ A-R
show False
by (auto simp add: Let-def)
qed
qed
qed
}
thus ?thesis
by (auto simp add: Let-def)
qed

{
fix j
fix p_j is_{sbj} $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_{sbj}$ j_j sb_j
assume j-bound: j < length ts_{sb}
assume ts_{sb}-j: ts_{sb}!j=(p_j,is_{sbj},j_j,sb_j,$\mathcal{D}_{sbj}$,$\mathcal{O}_j$,$\mathcal{R}_j$)
assume neq-i-j: i≠j
have A ∩ read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j) $\mathcal{O}_j$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j) = {}
proof −
{
let ?take-sb_j = (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)
let ?drop-sb_j = (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)

assume conflict: A ∩ read-only-reads (acquired True ?take-sb_j $\mathcal{O}_j$) ?drop-sb_j ≠ {}
have False
proof −
from conflict **obtain** a′ **where**
a′-in: a′ ∈ A **and**
a′-in-j: a′ ∈ read-only-reads (acquired True ?take-sb_j $\mathcal{O}_j$) ?drop-sb_j
by auto

from ts-sim [rule-format, OF j-bound] ts_{sb}-j j-bound
obtain p_j suspends$_j$ is_{sbj} $\mathcal{D}_{sbj}$ $\mathcal{D}_j$ j_{sbj} is_j **where**
ts_{sb}-j: ts_{sb} ! j = (p_j,is_{sbj}, j_{sbj}, sb_j,$\mathcal{D}_{sbj}$,$\mathcal{O}_j$,$\mathcal{R}_j$) **and**
suspends$_j$: suspends$_j$ = ?drop-sb_j **and**
is_j: instrs suspends$_j$ @ is_{sbj} = is_j @ prog-instrs suspends$_j$ **and**
$\mathcal{D}_j$: $\mathcal{D}_{sbj}$ = ($\mathcal{D}_j$ ∨ outstanding-refs is-volatile-Write$_{sb}$ sb_j ≠ {}) **and**
ts_j: ts!j = (hd-prog p_j suspends$_j$, is_j,
j_{sbj} |‘ (dom j_{sbj} − read-tmps suspends$_j$),(), $\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$,release
?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
apply (cases ts_{sb}!j)
apply (clarsimp simp add: Let-def)
done
from split-in-read-only-reads [OF a′-in-j [simplified suspends$_j$ [symmetric]]]
obtain t′ v′ ys zs **where**
split-suspends$_j$: suspends$_j$ = ys @ Read$_{sb}$ False a′ t′ v′# zs (**is** suspends$_j$ = ?suspends)
and
a′-unacq: a′ ∉ acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$)
by blast

from valid-program-history [OF j-bound ts_{sb}-j]
have causal-program-history is_{sbj} sb_j.
then have cph: causal-program-history is_{sbj} ?suspends
apply −
apply (rule causal-program-history-suffix [**where** sb=?take-sb_j])
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
apply (simp add: split-suspends$_j$)

done

from valid-last-prog [OF j-bound ts_{sb}-j] **have** last-prog: last-prog p_j sb_j = p_j.
then
have lp: last-prog p_j ?suspends = p_j
apply –
apply (rule last-prog-same-append [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

from valid-reads [OF j-bound ts_{sb}-j]
have reads-consis: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.
from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts_{sb}›
j-bound
ts_{sb}-j this]
have reads-consis-m-j: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
by (simp add: m $suspends_j$)

from outstanding-non-volatile-refs-owned-or-read-only [OF j-bound ts_{sb}-j]
have nvo-j: non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb_j.
with non-volatile-owned-or-read-only-append [of False $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb_j ?drop-sb_j]
have nvo-take-j: non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb_j
by auto

from a-unowned-others [rule-format, OF - neq-i-j] ts_{sb}-j j-bound
have a-not-acq: a $\notin$ acquired True ?take-sb_j $\mathcal{O}_j$
by auto

from a-notin-unforwarded-non-volatile-reads-drop[OF j-bound ts_{sb}-j neq-i-j]
have a-notin-unforwarded-reads: a $\notin$ unforwarded-non-volatile-reads $suspends_j$ {}
by (simp add: $suspends_j$)

let ?ma=(m(a := f (j_{sb}(t↦m a))))

from reads-consistent-mem-eq-on-unforwarded-non-volatile-reads [**where** W={}
and m′=?ma,simplified, OF - subset-refl reads-consis-m-j]
a-notin-unforwarded-reads
have reads-consis-ma-j:
reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) ?ma $suspends_j$
by auto

from reads-consis-ma-j
have reads-consis-ys: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) ?ma (ys)
by (simp add: split-$suspends_j$ reads-consistent-append)

from direct-memop-step.RMWWrite [**where** cond=cond **and** j=j_{sb} **and** m=m, OF cond′]
have (RMW a t (D, f) cond ret A L R W# is_{sb}′, j_{sb}, (), m, $\mathcal{D}$,$\mathcal{O}_{sb}$,$\mathcal{R}_{sb}$,$\mathcal{S}$) →

(is_{sb}', $j_{sb}(t \mapsto \text{ret } (m\ a)\ (f\ (j_{sb}(t \mapsto m\ a))))$, (), ?ma, False, $\mathcal{O}_{sb} \cup A - R$,Map.empty, $\mathcal{S} \oplus_W R \ominus_A L$).
from direct-computation.concurrent-step.Memop [OF i-bound′ ts-i this]
have step-a: (ts, m, $\mathcal{S}$) $\Rightarrow_d$
(ts[i := (p_{sb}, is_{sb}', $j_{sb}(t \mapsto \text{ret } (m\ a)\ (f\ (j_{sb}(t \mapsto m\ a))))$, (), False, $\mathcal{O}_{sb} \cup A - R$,Map.empty)],
?ma,$\mathcal{S} \oplus_W R \ominus_A L$)
(**is** - $\Rightarrow_d$ (?ts-a, -, ?shared-a)).

from ts_j neq-i-j j-bound

have ts-a-j: ?ts-a!j = (hd-prog p_j $suspends_j$, is_j,
j_{sbj} |‘ (dom j_{sbj} – read-tmps $suspends_j$),(), $\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$,release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
by auto

from valid-write-sops [OF j-bound ts_{sb}-j]
have ∀sop∈write-sops (?take-sb_j@?suspends). valid-sop sop
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain valid-sops-take: ∀sop∈write-sops ?take-sb_j. valid-sop sop **and**
valid-sops-drop: ∀sop∈write-sops (ys). valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound ts_{sb}-j]
have distinct-read-tmps (?take-sb_j@$suspends_j$)
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb_j ∩ read-tmps $suspends_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps $suspends_j$
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp only: distinct-read-tmps-append)
done

from valid-history [OF j-bound ts_{sb}-j]
have h-consis:
history-consistent j_{sbj} (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof –
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)

moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop h-consis] last-prog-hd-prog
have hist-consis′: history-consistent j_{sbj} (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis]
have no-vol-read: outstanding-refs is-volatile-$Read_{sb}$ (ys) = {}
by (auto simp add: outstanding-refs-append $suspends_j$ [symmetric]
split-$suspends_j$)

from j-bound leq **have** j-bound-ts-a: j < length ?ts-a **by** auto

from flush-store-buffer-append [**where** sb=ys **and** sb′=$Read_{sb}$ False a′ t′ v′#zs, simplified,
OF j-bound-ts-a is_j [simplified split-$suspends_j$] cph [simplified $suspends_j$]
ts-a-j [simplified split-$suspends_j$] refl lp [simplified split-$suspends_j$] reads-consis-ys
hist-consis′ [simplified split-$suspends_j$] valid-sops-drop
distinct-read-tmps-drop [simplified split-$suspends_j$]
no-volatile-$Read_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=?shared-a]

obtain is_j′ $\mathcal{R}_j$′ **where**
is_j′: Read False a′ t′# instrs zs @ is_{sbj} = is_j′ @ prog-instrs zs **and**
steps-ys: (?ts-a, ?ma, ?shared-a) $\Rightarrow_d^*$
(?ts-a[j:=(last-prog
(hd-prog p_j zs) ys,
is_j′,
j_{sbj} |\` (dom j_{sbj} − insert t′ (read-tmps zs)),
(), $\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ ys ≠ {}, acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$),$\mathcal{R}_j$′)],
flush ys (?ma),
share ys (?shared-a))
(**is** (-,-,-) $\Rightarrow_d^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto simp add: acquired-append)

from cph
have causal-program-history is_{sbj} ((ys @ [$Read_{sb}$ False a′ t′ v′]) @ zs)
by simp
from causal-program-history-suffix [OF this]

have cph′: causal-program-history is_{sbj} zs.
interpret $causal_j$: causal-program-history is_{sbj} zs **by** (rule cph′)

from $causal_j$.causal-program-history [of [], simplified, OF refl] is_j'
obtain is_j''
where is_j': $is_j' =$ Read False a′ t′#is_j'' **and**
is_j'': instrs zs @ $is_{sbj} = is_j''$ @ prog-instrs zs
by clarsimp

from i-bound′ **have** i-bound-ys: i < length ?ts-ys
by auto

from i-bound′ neq-i-j
have ts-ys-i: ?ts-ys!i = (p_{sb}, is_{sb}',
$j_{sb}(t \mapsto$ ret (m a) (f ($j_{sb}(t \mapsto$ m a)))),(), False, $\mathcal{O}_{sb} \cup A - R$,Map.empty)
by simp

from j-bound-ts-a **have** j-bound-ys: j < length ?ts-ys
by auto
then have ts-ys-j: ?ts-ys!j = (last-prog (hd-prog p_j zs) ys, Read False a′ t′#is_j'', j_{sbj}
|‘ (dom j_{sbj} − insert t′ (read-tmps zs)), (), $\mathcal{D}_j \vee$ outstanding-refs is-volatile-$Write_{sb}$ ys ≠
{},
acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$),$\mathcal{R}_j'$)
by (clarsimp simp add: is_j')
note conflict-computation = r-rtranclp-rtranclp [OF step-a steps-ys]

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys).

from safe-delayedE [OF this j-bound-ys ts-ys-j]

have a′ ∈ acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$) ∨
a′ ∈ read-only (share ys ($\mathcal{S} \oplus_W R \ominus_A L$))
apply cases
apply (auto simp add: Let-def is_{sb})
done

with a′-unacq
have a′-ro: a′ ∈ read-only (share ys ($\mathcal{S} \oplus_W R \ominus_A L$))
by auto
from a′-in
have a′-not-ro: a′ ∉ read-only ($\mathcal{S} \oplus_W R \ominus_A L$)
by (auto simp add: in-read-only-convs)

have a′ ∈ $\mathcal{O}_j$ ∪ all-acquired sb_j
proof −
{
assume a-notin: a′ ∉ $\mathcal{O}_j$ ∪ all-acquired sb_j
from weak-sharing-consis [OF j-bound ts_{sb}-j]

```
have weak-sharing-consistent 𝒪j sbj.
with weak-sharing-consistent-append [of 𝒪j ?take-sbj ?drop-sbj]
have weak-sharing-consistent (acquired True ?take-sbj 𝒪j) suspendsj
  by (auto simp add: suspendsj)
with split-suspendsj
have weak-consis: weak-sharing-consistent (acquired True ?take-sbj 𝒪j) ys
  by (simp add: weak-sharing-consistent-append)
from all-acquired-append [of ?take-sbj ?drop-sbj]
have all-acquired ys ⊆ all-acquired sbj
  apply (clarsimp)
 apply (clarsimp simp add: suspendsj [symmetric] split-suspendsj all-acquired-append)
  done
     with a-notin acquired-takeWhile-non-volatile-Writesb [of sbj 𝒪j]
          all-acquired-append [of ?take-sbj ?drop-sbj]
have a′ ∉ acquired True (takeWhile (Not ∘ is-volatile-Writesb) sbj) 𝒪j ∪ all-acquired
ys
          by auto

from read-only-share-unowned [OF weak-consis this a′-ro]
have a′ ∈ read-only (𝒮 ⊕W R ⊖A L) .

with a′-not-ro have False
  by auto
with a-notin read-only-share-unowned [OF weak-consis - a′-ro]
  all-acquired-takeWhile [of (Not ∘ is-volatile-Writesb) sbj]

have a′ ∈ read-only (𝒮 ⊕W R ⊖A L)
  by (auto simp add: acquired-takeWhile-non-volatile-Writesb)
with a′-not-ro have False
  by auto
}
thus ?thesis by blast
  qed

  moreover
  from A-unacquired-by-others [rule-format, OF - neq-i-j] tssb-j j-bound
  have A ∩ all-acquired sbj = {}
by (auto simp add: Let-def)
  moreover
  from A-unowned-by-others [rule-format, OF - neq-i-j] tssb-j j-bound
  have A ∩ 𝒪j = {}
    by (auto simp add: Let-def dest: all-shared-acquired-in)
  moreover note a′-in
  ultimately
  show False
by auto
 qed
}
thus ?thesis
 by (auto simp add: Let-def)
```

qed
} **note** A-no-read-only-reads = this

have valid-own′: valid-ownership $\mathcal{S}_{\mathsf{sb}}{}'$ $\mathrm{ts}_{\mathsf{sb}}{}'$
proof (intro-locales)
show outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}_{\mathsf{sb}}{}'$ $\mathrm{ts}_{\mathsf{sb}}{}'$
proof
fix j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j p_j
assume j-bound: j < length $\mathrm{ts}_{\mathsf{sb}}{}'$
assume $\mathrm{ts}_{\mathsf{sb}}{}'$-j: $\mathrm{ts}_{\mathsf{sb}}{}'$!j = ($\mathrm{p}_j$,$\mathrm{is}_j$,$\mathrm{j}_j$,$\mathrm{sb}_j$,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show non-volatile-owned-or-read-only False $\mathcal{S}_{\mathsf{sb}}{}'$ $\mathcal{O}_j$ sb_j
proof (cases j=i)
case True
have non-volatile-owned-or-read-only False
($\mathcal{S}_{\mathsf{sb}}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}_{\mathsf{sb}}$ ∪ A − R) []
by simp
then show ?thesis
using True i-bound $\mathrm{ts}_{\mathsf{sb}}{}'$-j
by (auto simp add: $\mathrm{ts}_{\mathsf{sb}}{}'$ $\mathcal{S}_{\mathsf{sb}}{}'$ sb sb′)
next
case False
from j-bound **have** j-bound′: j < length $\mathrm{ts}_{\mathsf{sb}}$
by (auto simp add: $\mathrm{ts}_{\mathsf{sb}}{}'$)
with $\mathrm{ts}_{\mathsf{sb}}{}'$-j False i-bound
have $\mathrm{ts}_{\mathsf{sb}}$-j: $\mathrm{ts}_{\mathsf{sb}}$!j = ($\mathrm{p}_j$,$\mathrm{is}_j$,$\mathrm{j}_j$,$\mathrm{sb}_j$,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (auto simp add: $\mathrm{ts}_{\mathsf{sb}}{}'$)

note nvo = outstanding-non-volatile-refs-owned-or-read-only [OF j-bound′ $\mathrm{ts}_{\mathsf{sb}}$-j]

from read-only-unowned [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i] R-owned
have R ∩ read-only $\mathcal{S}_{\mathsf{sb}}$ = {}
by auto
with A-no-read-only-reads [OF j-bound′ $\mathrm{ts}_{\mathsf{sb}}$-j False [symmetric]] L-subset
have ∀ a∈read-only-reads
(acquired True (takeWhile (Not ∘ is-volatile-$\mathrm{Write}_{\mathsf{sb}}$) sb_j) $\mathcal{O}_j$)
(dropWhile (Not ∘ is-volatile-$\mathrm{Write}_{\mathsf{sb}}$) sb_j).
a ∈ read-only $\mathcal{S}_{\mathsf{sb}}$ ⟶ a ∈ read-only ($\mathcal{S}_{\mathsf{sb}}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (auto simp add: in-read-only-convs)
from non-volatile-owned-or-read-only-read-only-reads-eq′ [OF nvo this]
have non-volatile-owned-or-read-only False ($\mathcal{S}_{\mathsf{sb}}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $\mathcal{O}_j$ sb_j.
thus ?thesis **by** (simp add: $\mathcal{S}_{\mathsf{sb}}{}'$)
qed
qed
next
show outstanding-volatile-writes-unowned-by-others $\mathrm{ts}_{\mathsf{sb}}{}'$
proof (unfold-locales)
fix i_1 j p_1 is_1 $\mathcal{O}_1$ $\mathcal{R}_1$ $\mathcal{D}_1$ xs_1 sb_1 p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs_j sb_j
assume i_1-bound: i_1 < length $\mathrm{ts}_{\mathsf{sb}}{}'$
assume j-bound: j < length $\mathrm{ts}_{\mathsf{sb}}{}'$

assume i_1-j: $i_1 \neq j$
assume ts-i_1: $ts_{sb}'!i_1 = (p_1,is_1,xs_1,sb_1,\mathcal{D}_1,\mathcal{O}_1,\mathcal{R}_1)$
assume ts-j: $ts_{sb}'!j = (p_j,is_j,xs_j,sb_j,\mathcal{D}_j,\mathcal{O}_j,\mathcal{R}_j)$
show $(\mathcal{O}_j \cup$ all-acquired $sb_j) \cap$ outstanding-refs is-volatile-Write$_{sb}$ $sb_1 = \{\}$
proof (cases i_1=i)
case True
with ts-i_1 i-bound **show** ?thesis
by (simp add: ts_{sb}' sb$'$ sb)
next
case False
note i_1-i = this
from i_1-bound **have** i_1-bound$'$: $i_1 <$ length ts_{sb}
by (simp add: ts_{sb}' sb$'$ sb)
hence i_1-bound$''$: $i_1 <$ length (map owned ts_{sb})
by auto
from ts-i_1 False **have** ts-i_1': $ts_{sb}!i_1 = (p_1,is_1,xs_1,sb_1,\mathcal{D}_1,\mathcal{O}_1,\mathcal{R}_1)$
by (simp add: ts_{sb}' sb$'$ sb)
show ?thesis
proof (cases j=i)
case True

from i-bound ts-j ts_{sb}' True **have** sb_j: sb_j=[]
by (simp add: ts_{sb}' sb$'$)
from A-unused-by-others [rule-format, OF - False [symmetric]] ts-i_1 i_1-bound$''$
False i_1-bound$'$
have A $\cap$ ($\mathcal{O}_1 \cup$ outstanding-refs is-volatile-Write$_{sb}$ sb_1) = {}
by (auto simp add: Let-def ts_{sb}' $\mathcal{O}_{sb}'$ sb$'$ owned-def)
moreover
from outstanding-volatile-writes-unowned-by-others
[OF i_1-bound$'$ i-bound i_1-i ts-i_1' ts_{sb}-i]
have $\mathcal{O}_{sb} \cap$ outstanding-refs is-volatile-Write$_{sb}$ $sb_1 = \{\}$ **by** (simp add: sb)

ultimately show ?thesis **using** ts-j True
by (auto simp add: i-bound ts_{sb}' $\mathcal{O}_{sb}'$ sb_j)
next
case False
from j-bound **have** j-bound$'$: j < length ts_{sb}
by (simp add: ts_{sb}')
from ts-j False **have** ts-j$'$: $ts_{sb}!j = (p_j,is_j,xs_j,sb_j,\mathcal{D}_j,\mathcal{O}_j,\mathcal{R}_j)$
by (simp add: ts_{sb}')
from outstanding-volatile-writes-unowned-by-others
[OF i_1-bound$'$ j-bound$'$ i_1-j ts-i_1' ts-j$'$]
show $(\mathcal{O}_j \cup$ all-acquired $sb_j) \cap$ outstanding-refs is-volatile-Write$_{sb}$ $sb_1 = \{\}$ **.**
qed
qed
qed
next
show read-only-reads-unowned ts_{sb}'
proof
fix n m

fix p_n is_n $\mathcal{O}_n$ $\mathcal{R}_n$ $\mathcal{D}_n$ j_n sb_n p_m is_m $\mathcal{O}_m$ $\mathcal{R}_m$ $\mathcal{D}_m$ j_m sb_m
assume n-bound: n < length ts_{sb}'
and m-bound: m < length ts_{sb}'
and neq-n-m: n≠m
and nth: ts_{sb}'!n = (p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
and mth: ts_{sb}'!m =(p_m, is_m, j_m, sb_m, $\mathcal{D}_m$, $\mathcal{O}_m$,$\mathcal{R}_m$)
from n-bound **have** n-bound': n < length ts_{sb} **by** (simp add: ts_{sb}')
from m-bound **have** m-bound': m < length ts_{sb} **by** (simp add: ts_{sb}')
show ($\mathcal{O}_m$ ∪ all-acquired sb_m) ∩
read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n) $\mathcal{O}_n$)
(dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n) =
{}
proof (cases m=i)
case True
with neq-n-m **have** neq-n-i: n≠i
by auto

with n-bound nth i-bound **have** nth': ts_{sb}!n =(p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
by (auto simp add: ts_{sb}')
note read-only-reads-unowned [OF n-bound' i-bound neq-n-i nth' ts_{sb}-i]
moreover
note A-no-read-only-reads [OF n-bound' nth']
ultimately
show ?thesis
using True ts_{sb}-i neq-n-i nth mth n-bound' m-bound'
by (auto simp add: ts_{sb}' $\mathcal{O}_{sb}'$ sb sb')
next
case False
note neq-m-i = this
with m-bound mth i-bound **have** mth': ts_{sb}!m = (p_m, is_m, j_m, sb_m, $\mathcal{D}_m$, $\mathcal{O}_m$,$\mathcal{R}_m$)
by (auto simp add: ts_{sb}')
show ?thesis
proof (cases n=i)
case True
with ts_{sb}-i nth mth neq-m-i n-bound'
show ?thesis
by (auto simp add: ts_{sb}' sb')
next
case False
with n-bound nth i-bound **have** nth': ts_{sb}!n =(p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
by (auto simp add: ts_{sb}')
from read-only-reads-unowned [OF n-bound' m-bound' neq-n-m nth' mth'] False
neq-m-i
show ?thesis
by (clarsimp)
qed
qed
qed
next
show ownership-distinct ts_{sb}'

proof (unfold-locales)
  fix i_1 j p_1 is_1 $\mathcal{O}_1$ $\mathcal{R}_1$ $\mathcal{D}_1$ xs_1 sb_1 p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs_j sb_j
  assume i_1-bound: i_1 < length ts_{sb}'
  assume j-bound: j < length ts_{sb}'
  assume i_1-j: $i_1 \neq j$
  assume ts-i_1: $ts_{sb}'!i_1 = (p_1,is_1,xs_1,sb_1,\mathcal{D}_1,\mathcal{O}_1,\mathcal{R}_1)$
  assume ts-j: $ts_{sb}'!j = (p_j,is_j,xs_j,sb_j,\mathcal{D}_j,\mathcal{O}_j,\mathcal{R}_j)$
  show ($\mathcal{O}_1$ ∪ all-acquired sb_1) ∩ ($\mathcal{O}_j$ ∪ all-acquired sb_j)= {}
  proof (cases i_1=i)
    case True
    with i_1-j **have** i-j: i≠j
      by simp

    from i-bound ts-i_1 ts_{sb}' True **have** sb_1: sb_1=[]
      by (simp add: ts_{sb}' sb′)
    from j-bound **have** j-bound′: j < length ts_{sb}
      by (simp add: ts_{sb}')
    hence j-bound″: j < length (map owned ts_{sb})
      by simp
    from ts-j i-j **have** ts-j′: $ts_{sb}!j = (p_j,is_j,xs_j,sb_j,\mathcal{D}_j,\mathcal{O}_j,\mathcal{R}_j)$
      by (simp add: ts_{sb}')

    from A-unused-by-others [rule-format, OF - i-j] ts-j i-j j-bound′
    have A ∩ ($\mathcal{O}_j$ ∪ outstanding-refs is-volatile-$Write_{sb}$ sb_j) = {}
      by (auto simp add: Let-def ts_{sb}' owned-def)
    moreover
    from A-unacquired-by-others [rule-format, OF - i-j] ts-j i-j j-bound′
    have A ∩ all-acquired sb_j = {}
      by (auto simp add: Let-def ts_{sb}')

    moreover
    from ownership-distinct [OF i-bound j-bound′ i-j ts_{sb}-i ts-j′]
    have $\mathcal{O}_{sb}$ ∩ ($\mathcal{O}_j$ ∪ all-acquired sb_j)= {} **by** (simp add: sb)
    ultimately show ?thesis **using** ts-i_1 True
      by (auto simp add: i-bound ts_{sb}' $\mathcal{O}_{sb}'$ sb′ sb_1)
  next
    case False
    note i_1-i = this
    from i_1-bound **have** i_1-bound′: i_1 < length ts_{sb}
      by (simp add: ts_{sb}')
    hence i_1-bound″: i_1 < length (map owned ts_{sb})
      by simp
    from ts-i_1 False **have** ts-i_1′: $ts_{sb}!i_1 = (p_1,is_1,xs_1,sb_1,\mathcal{D}_1,\mathcal{O}_1,\mathcal{R}_1)$
      by (simp add: ts_{sb}')
    show ?thesis
    proof (cases j=i)
      case True
      from A-unused-by-others [rule-format, OF - False [symmetric]] ts-i_1
 False i_1-bound′
      have A ∩ ($\mathcal{O}_1$ ∪ outstanding-refs is-volatile-$Write_{sb}$ sb_1) = {}

by (auto simp add: Let-def ts_{sb}' owned-def)
moreover
from A-unacquired-by-others [rule-format, OF - False [symmetric]] ts-i_1 False i_1-bound$'$
have A ∩ all-acquired sb_1 = {}
by (auto simp add: Let-def ts_{sb}' owned-def)
moreover
from ownership-distinct [OF i_1-bound$'$ i-bound i_1-i ts-i_1' ts_{sb}-i]
have ($\mathcal{O}_1$ ∪ all-acquired sb_1) ∩ $\mathcal{O}_{sb}$ = {} **by** (simp add: sb)
ultimately show ?thesis
using ts-j True
by (auto simp add: i-bound ts_{sb}' $\mathcal{O}_{sb}'$ sb$'$)
next
case False
from j-bound **have** j-bound$'$: j < length ts_{sb}
by (simp add: ts_{sb}')
from ts-j False **have** ts-j$'$: ts_{sb}!j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (simp add: ts_{sb}')
from ownership-distinct [OF i_1-bound$'$ j-bound$'$ i_1-j ts-i_1' ts-j$'$]
show ($\mathcal{O}_1$ ∪ all-acquired sb_1) ∩ ($\mathcal{O}_j$ ∪ all-acquired sb_j) = {} **.**
qed
qed
qed
qed

have valid-hist$'$: valid-history program-step ts_{sb}'
proof −
from valid-history [OF i-bound ts_{sb}-i]
have history-consistent (j_{sb}(t↦ret (m_{sb} a) (f (j_{sb}(t↦m_{sb} a))))) (hd-prog p_{sb} []) [] **by** simp
from valid-history-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' j_{sb}' sb$'$ sb)
qed

from valid-reads [OF i-bound ts_{sb}-i]
have reads-consis: reads-consistent False $\mathcal{O}_{sb}$ m_{sb} sb **.**

have valid-reads$'$: valid-reads m_{sb}' ts_{sb}'
proof (unfold-locales)
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ acq_j j_j sb_j
assume j-bound: j < length ts_{sb}'
assume ts-j: ts_{sb}'!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show reads-consistent False $\mathcal{O}_j$ m_{sb}' sb_j
proof (cases i=j)
case True
from reads-consis ts-j j-bound sb **show** ?thesis
by (clarsimp simp add: True m_{sb}' $Write_{sb}$ ts_{sb}' sb$'$)
next
case False
from j-bound **have** j-bound$'$: j < length ts_{sb}
by (simp add: ts_{sb}')

moreover from ts-j False **have** ts-j′: ts_{sb} ! j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
using j-bound **by** (simp add: ts_{sb}′)
ultimately have consis-m: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j
by (rule valid-reads)
let ?m′ = (m_{sb}(a := f (j_{sb}(t ↦ m_{sb} a))))
from a-unowned-others [rule-format, OF - False] j-bound′ ts-j′
obtain a-acq: a ∉ acquired True (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j) $\mathcal{O}_j$ **and**
a-unsh: a ∉ all-shared (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)
by auto
with a-notin-unforwarded-non-volatile-reads-drop [OF j-bound′ ts-j′ False]
have ∀a∈acquired True (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j) $\mathcal{O}_j$ ∪
all-shared (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j) ∪
unforwarded-non-volatile-reads (dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j) {}.
?m′ a = m_{sb} a
by auto
from reads-consistent-mem-eq-on-unforwarded-non-volatile-reads-drop
[**where** W={},simplified, OF this - - consis-m]
acquired-reads-all-acquired′ [of (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j) $\mathcal{O}_j$]
have reads-consistent False $\mathcal{O}_j$ (m_{sb}(a := f (j_{sb}(t ↦ m_{sb} a)))) sb_j
by (auto simp del: fun-upd-apply)
thus ?thesis
by (simp add: m_{sb}′)
qed
qed

have valid-sharing′: valid-sharing ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) ts_{sb}′
proof (intro-locales)
show outstanding-non-volatile-writes-unshared ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) ts_{sb}′
proof (unfold-locales)
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ acq_j xs_j sb_j
assume j-bound: j < length ts_{sb}′
assume jth: ts_{sb}′ ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show non-volatile-writes-unshared ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) sb_j
proof (cases i=j)
case True
with i-bound jth **show** ?thesis
by (simp add: ts_{sb}′ sb′ sb)
next
case False
from j-bound **have** j-bound′: j < length ts_{sb}
by (auto simp add: ts_{sb}′)
from jth False **have** jth′: ts_{sb} ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (auto simp add: ts_{sb}′)
from outstanding-non-volatile-writes-unshared [OF j-bound′ jth′]
have unshared: non-volatile-writes-unshared $\mathcal{S}_{sb}$ sb_j.
have ∀a∈dom ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) − dom $\mathcal{S}_{sb}$. a ∉ outstanding-refs is-non-volatile-$Write_{sb}$ sb_j
proof −
{
fix a

assume a-in: a ∈ dom ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) − dom $\mathcal{S}_{sb}$
hence a-R: a ∈ R
by clarsimp
assume a-in-j: a ∈ outstanding-refs is-non-volatile-Write$_{sb}$ sb$_j$
have False
proof −
from non-volatile-owned-or-read-only-outstanding-non-volatile-writes [OF outstanding-non-volatile-refs-owned-or-read-only [OF j-bound′ jth′]]
a-in-j
have a ∈ $\mathcal{O}_j$ ∪ all-acquired sb$_j$
by auto
moreover
with ownership-distinct [OF i-bound j-bound′ False ts$_{sb}$-i jth′] a-R R-owned
show False
by blast
qed
}
thus ?thesis **by** blast
qed

from non-volatile-writes-unshared-no-outstanding-non-volatile-Write$_{sb}$
[OF unshared this]
show ?thesis **.**
qed
qed
next
show sharing-consis ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) ts$_{sb}$′
proof (unfold-locales)
fix j p$_j$ is$_j$ $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ acq$_j$ xs$_j$ sb$_j$
assume j-bound: j < length ts$_{sb}$′
assume jth: ts$_{sb}$′ ! j = (p$_j$,is$_j$,xs$_j$,sb$_j$,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show sharing-consistent ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) $\mathcal{O}_j$ sb$_j$
proof (cases i=j)
case True
with i-bound jth **show** ?thesis
by (simp add: ts$_{sb}$′ sb′ sb)
next
case False
from j-bound **have** j-bound′: j < length ts$_{sb}$
by (auto simp add: ts$_{sb}$′)
from jth False **have** jth′: ts$_{sb}$! j = (p$_j$,is$_j$,xs$_j$,sb$_j$,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (auto simp add: ts$_{sb}$′)
from sharing-consis [OF j-bound′ jth′]
have consis: sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb$_j$**.**

have acq-cond: all-acquired sb$_j$ ∩ dom $\mathcal{S}_{sb}$ − dom ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) = {}
proof −
{
fix a
assume a-acq: a ∈ all-acquired sb$_j$

assume a ∈ dom $\mathcal{S}_{\mathsf{sb}}$
assume a-L: a ∈ L
have False
proof −
from A-unacquired-by-others [rule-format, of j,OF - False] j-bound′ jth′
have A ∩ all-acquired sb_j = {}
by auto
with a-acq a-L L-subset
show False
by blast
qed
}
thus ?thesis
by auto
qed
have uns-cond: all-unshared sb_j ∩ dom ($\mathcal{S}_{\mathsf{sb}}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) − dom $\mathcal{S}_{\mathsf{sb}}$ = {}
proof −
{
fix a
assume a-uns: a ∈ all-unshared sb_j
assume a ∉ L
assume a-R: a ∈ R
have False
proof −
from unshared-acquired-or-owned [OF consis] a-uns
have a ∈ all-acquired sb_j ∪ $\mathcal{O}_j$ **by** auto
with ownership-distinct [OF i-bound j-bound′ False ts_{sb}-i jth′] R-owned a-R
show False
by blast
qed
}
thus ?thesis
by auto
qed

from sharing-consistent-preservation [OF consis acq-cond uns-cond]
show ?thesis
by (simp add: ts_{sb}′)
qed
qed
next
show unowned-shared ($\mathcal{S}_{\mathsf{sb}}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ts_{sb}′
proof (unfold-locales)
show − $\bigcup$((λ(-,-, -, -,-, $\mathcal{O}$,-). $\mathcal{O}$) ‘ set ts_{sb}′) ⊆ dom ($\mathcal{S}_{\mathsf{sb}}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
proof −

have s: $\bigcup$((λ(-,-, -, -,-, $\mathcal{O}$,-). $\mathcal{O}$) ‘ set ts_{sb}′) =
$\bigcup$((λ(-,-, -, -,-, $\mathcal{O}$,-). $\mathcal{O}$) ‘ set ts_{sb}) ∪ A − R

apply (unfold ts_{sb}′ $\mathcal{O}_{\mathsf{sb}}$′)

apply (rule acquire-release-ownership-nth-update [OF R-owned i-bound ts_{sb}-i])
apply fact
done

note unowned-shared L-subset A-R
then
show ?thesis
apply (simp only: s)
apply auto
done
qed
qed
next
show read-only-unowned ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) ts_{sb}'
proof
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ acq_j xs_j sb_j
assume j-bound: j < length ts_{sb}'
assume jth: ts_{sb}' ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show $\mathcal{O}_j$ ∩ read-only ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) = {}
proof (cases i=j)
case True
from read-only-unowned [OF i-bound ts_{sb}-i] R-owned A-R
have ($\mathcal{O}_{sb}$ ∪ A − R) ∩ read-only ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) = {}
by (auto simp add: in-read-only-convs)
with jth ts_{sb}-i i-bound True
show ?thesis
by (auto simp add: $\mathcal{O}_{sb}'$ ts_{sb}')
next
case False
from j-bound **have** j-bound′: j < length ts_{sb}
by (auto simp add: ts_{sb}')
with False jth **have** jth′: ts_{sb} ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (auto simp add: ts_{sb}')
from read-only-unowned [OF j-bound′ jth′]
have $\mathcal{O}_j$ ∩ read-only $\mathcal{S}_{sb}$ = {}.
moreover
from A-unowned-by-others [rule-format, OF - False] j-bound′ jth′
have A ∩ $\mathcal{O}_j$ = {}
by (auto dest: all-shared-acquired-in)
moreover
from ownership-distinct [OF i-bound j-bound′ False ts_{sb}-i jth′]
have $\mathcal{O}_{sb}$ ∩ $\mathcal{O}_j$ = {}
by auto
moreover note R-owned A-R
ultimately show ?thesis
by (fastforce simp add: in-read-only-convs split: if-split-asm)
qed
qed
next
show no-outstanding-write-to-read-only-memory ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) ts_{sb}'

proof
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ acq_j xs_j sb_j
assume j-bound: j < length ts_{sb}'
assume jth: ts_{sb}' ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show no-write-to-read-only-memory ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) sb_j
proof (cases i=j)
case True
with jth ts_{sb}-i i-bound
show ?thesis
by (auto simp add: sb sb′ ts_{sb}')
next
case False
from j-bound **have** j-bound′: j < length ts_{sb}
by (auto simp add: ts_{sb}')
with False jth **have** jth′: ts_{sb} ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (auto simp add: ts_{sb}')
from no-outstanding-write-to-read-only-memory [OF j-bound′ jth′]
have nw: no-write-to-read-only-memory $\mathcal{S}_{sb}$ sb_j.
have R ∩ outstanding-refs is-$Write_{sb}$ sb_j = {}
proof −
note dist = ownership-distinct [OF i-bound j-bound′ False ts_{sb}-i jth′]
from non-volatile-owned-or-read-only-outstanding-non-volatile-writes
[OF outstanding-non-volatile-refs-owned-or-read-only [OF j-bound′ jth′]]
dist
have outstanding-refs is-non-volatile-$Write_{sb}$ sb_j ∩ $\mathcal{O}_{sb}$ = {}
by auto
moreover
from outstanding-volatile-writes-unowned-by-others [OF j-bound′ i-bound
False [symmetric] jth′ ts_{sb}-i]
have outstanding-refs is-volatile-$Write_{sb}$ sb_j ∩ $\mathcal{O}_{sb}$ = {}
by auto
ultimately have outstanding-refs is-$Write_{sb}$ sb_j ∩ $\mathcal{O}_{sb}$ = {}
by (auto simp add: misc-outstanding-refs-convs)
with R-owned
show ?thesis **by** blast
qed
then
have ∀a∈outstanding-refs is-$Write_{sb}$ sb_j.
a ∈ read-only ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) ⟶ a ∈ read-only $\mathcal{S}_{sb}$
by (auto simp add: in-read-only-convs)

from no-write-to-read-only-memory-read-only-reads-eq [OF nw this]
show ?thesis .
qed
qed
qed

have tmps-distinct′: tmps-distinct ts_{sb}'
proof (intro-locales)
from load-tmps-distinct [OF i-bound ts_{sb}-i]

have distinct-load-tmps is_{sb}'
 by (auto simp add: is_{sb} split: instr.splits)
from load-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$ is_{sb})
 next
from read-tmps-distinct [OF i-bound ts_{sb}-i]
have distinct-read-tmps [] **by** (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$)
from read-tmps-distinct-nth-update [OF i-bound this]
show read-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$)
 next
from load-tmps-read-tmps-distinct [OF i-bound ts_{sb}-i]
 load-tmps-distinct [OF i-bound ts_{sb}-i]
have load-tmps is_{sb}' $\cap$ read-tmps [] = {}
 by (clarsimp)
from load-tmps-read-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-read-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$)
 qed

 have valid-sops′: valid-sops ts_{sb}'
 proof −
from valid-store-sops [OF i-bound ts_{sb}-i]
obtain
 valid-store-sops′: $\forall$sop$\in$store-sops is_{sb}'. valid-sop sop
 by (auto simp add: is_{sb} ts_{sb}' sb' sb $\mathcal{O}_{sb}'$)

from valid-sops-nth-update [OF i-bound - valid-store-sops′, **where** sb= []]
show ?thesis **by** (auto simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$)
 qed

 have valid-dd′: valid-data-dependency ts_{sb}'
 proof −
from data-dependency-consistent-instrs [OF i-bound ts_{sb}-i]
obtain
 dd-is: data-dependency-consistent-instrs (dom j_{sb}') is_{sb}'
 by (auto simp add: is_{sb} j_{sb}')
from load-tmps-write-tmps-distinct [OF i-bound ts_{sb}-i]
have load-tmps is_{sb}' $\cap$ $\bigcup$(fst ‘ write-sops []) = {}
 by (auto simp add: write-sops-append)
from valid-data-dependency-nth-update [OF i-bound dd-is this]
show ?thesis **by** (simp add: ts_{sb}' sb' sb $\mathcal{O}_{sb}'$)
 qed

 have load-tmps-fresh′: load-tmps-fresh ts_{sb}'
 proof −
from load-tmps-fresh [OF i-bound ts_{sb}-i]
have load-tmps (RMW a t (D,f) cond ret A L R W # is_{sb}') $\cap$ dom j_{sb} = {}
 by (simp add: is_{sb})
moreover
from load-tmps-distinct [OF i-bound ts_{sb}-i] **have** t $\notin$ load-tmps is_{sb}'
 by (auto simp add: is_{sb})

ultimately have load-tmps $is_{sb}' \cap$ dom (j_{sb}(t $\mapsto$ ret (m_{sb} a) (f (j_{sb}(t$\mapsto$$m_{sb}$ a))))) = {}
by auto
from load-tmps-fresh-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' sb' j_{sb}')
qed

from enough-flushs-nth-update [OF i-bound, **where** sb=[]]
have enough-flushs': enough-flushs ts_{sb}'
by (auto simp: ts_{sb}' sb' sb)

have valid-program-history': valid-program-history ts_{sb}'
proof –
have causal': causal-program-history is_{sb}' sb'
by (simp add: is_{sb} sb sb')
have last-prog p_{sb} sb' = p_{sb}
by (simp add: sb' sb)
from valid-program-history-nth-update [OF i-bound causal' this]
show ?thesis
by (simp add: ts_{sb}' sb')
qed

from is-sim **have** is: is = RMW a t (D,f) cond ret A L R W # is_{sb}'
by (simp add: suspends sb is_{sb})

from direct-memop-step.RMWWrite [**where** cond=cond **and** j=j_{sb} **and** m=m, OF cond']
have (RMW a t (D, f) cond ret A L R W # is_{sb}', j_{sb}, (),m, $\mathcal{D}$, $\mathcal{O}_{sb}$,$\mathcal{R}_{sb}$, $\mathcal{S}$) $\rightarrow$
(is_{sb}',j_{sb}(t $\mapsto$ ret (m a) (f (j_{sb}(t$\mapsto$m a)))), (),
m(a := f (j_{sb}(t $\mapsto$ m a))), False, $\mathcal{O}_{sb}$ $\cup$ A – R, Map.empty, $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L).

from direct-computation.concurrent-step.Memop [OF i-bound' ts-i this]
have (ts, m, $\mathcal{S}$) $\Rightarrow_d$ (ts[i := (p_{sb}, is_{sb}',j_{sb}(t $\mapsto$ ret (m a) (f (j_{sb}(t$\mapsto$m a)))), (), False, $\mathcal{O}_{sb}$ $\cup$ A – R,Map.empty)],
m(a := f (j_{sb}(t $\mapsto$ m a))),$\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L).

moreover

have tmps-commute: j_{sb}(t $\mapsto$ ret (m_{sb} a) (f (j_{sb}(t$\mapsto$$m_{sb}$ a)))) =
(j_{sb} |` (dom j_{sb} – {t}))(t $\mapsto$ ret (m_{sb} a) (f (j_{sb}(t$\mapsto$$m_{sb}$ a))))
apply (rule ext)
apply (auto simp add: restrict-map-def domIff)
done

from a-unflushed ts_{sb}-i sb
have a-unflushed':
$\forall$j < length ts_{sb}.
(let (-,-,-,sb_j,-,-,-) = ts_{sb}!j

in a $\notin$ outstanding-refs is-non-volatile-Write$_{sb}$ (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_j$))
by auto

have all-shared-L: $\forall$ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ acq j sb. i < length ts$_{sb}$ $\longrightarrow$
ts$_{sb}$! i = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\cap$ L = {}
proof –
{
fix j p$_j$ is$_j$ $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j$_j$ sb$_j$ x
assume j-bound: j < length ts$_{sb}$
assume jth: ts$_{sb}$!j = (p$_j$,is$_j$,j$_j$,sb$_j$,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
assume x-shared: x $\in$ all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_j$)
assume x-L: x $\in$ L
have False
proof (cases i=j)
case True **with** x-shared ts$_{sb}$-i jth **show** False **by** (simp add: sb)
next
case False
show False
proof –
from all-shared-acquired-or-owned [OF sharing-consis [OF j-bound jth]]
have all-shared sb$_j$ $\subseteq$ all-acquired sb$_j$ $\cup$ $\mathcal{O}_j$.

moreover have all-shared (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_j$) $\subseteq$ all-shared sb$_j$
using all-shared-append [of (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_j$)
(dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb$_j$)]
by auto
moreover
from A-unacquired-by-others [rule-format, OF - False] jth j-bound
have A $\cap$ all-acquired sb$_j$ = {} **by** auto
moreover

from A-unowned-by-others [rule-format, OF - False] jth j-bound
have A $\cap$ $\mathcal{O}_j$ = {}
by (auto dest: all-shared-acquired-in)

ultimately
show False
using L-subset x-L x-shared
by blast
qed
qed
}
thus ?thesis **by** blast
qed

have all-shared-A: $\forall$ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts$_{sb}$ $\longrightarrow$
ts$_{sb}$! i = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$

all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ A = {}
proof –
{
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j x
assume j-bound: j < length ts_{sb}
assume jth: ts_{sb}!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
assume x-shared: x ∈ all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)
assume x-A: x ∈ A
have False
proof (cases i=j)
case True **with** x-shared ts_{sb}-i jth **show** False **by** (simp add: sb)
next
case False
show False
proof –
from all-shared-acquired-or-owned [OF sharing-consis [OF j-bound jth]]
have all-shared sb_j ⊆ all-acquired sb_j ∪ $\mathcal{O}_j$.

moreover have all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j) ⊆ all-shared sb_j
using all-shared-append [of (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)]
by auto
moreover
from A-unacquired-by-others [rule-format, OF - False] jth j-bound
have A ∩ all-acquired sb_j = {} **by** auto
moreover

from A-unowned-by-others [rule-format, OF - False] jth j-bound
have A ∩ $\mathcal{O}_j$ = {}
by (auto dest: all-shared-acquired-in)

ultimately
show False
using x-A x-shared
by blast
qed
qed
}
thus ?thesis **by** blast
qed
hence all-shared-L: ∀ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts_{sb} ⟶
ts_{sb} ! i = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) ⟶
all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ L = {}
using L-subset **by** blast

have all-unshared-R: ∀ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts_{sb} ⟶
ts_{sb} ! i = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) ⟶
all-unshared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {}

proof −
{
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j x
assume j-bound: j < length ts_{sb}
assume jth: $ts_{sb}!j = (p_j,is_j,j_j,sb_j,\mathcal{D}_j,\mathcal{O}_j,\mathcal{R}_j)$
assume x-unshared: x ∈ all-unshared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)
assume x-R: x ∈ R
have False
proof (cases i=j)
case True **with** x-unshared ts_{sb}-i jth **show** False **by** (simp add: sb)
next
case False
show False
proof −
from unshared-acquired-or-owned [OF sharing-consis [OF j-bound jth]]
have all-unshared sb_j ⊆ all-acquired sb_j ∪ $\mathcal{O}_j$.

moreover have all-unshared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j) ⊆ all-unshared sb_j
using all-unshared-append [of (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)]
by auto
moreover

note ownership-distinct [OF i-bound j-bound False ts_{sb}-i jth]

ultimately
show False
using R-owned x-R x-unshared
by blast
qed
qed
}
thus ?thesis **by** blast
qed

have all-acquired-R: ∀ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts_{sb} ⟶
ts_{sb} ! i = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) ⟶
all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {}
proof −
{
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j x
assume j-bound: j < length ts_{sb}
assume jth: $ts_{sb}!j = (p_j,is_j,j_j,sb_j,\mathcal{D}_j,\mathcal{O}_j,\mathcal{R}_j)$
assume x-acq: x ∈ all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)
assume x-R: x ∈ R
have False
proof (cases i=j)
case True **with** x-acq ts_{sb}-i jth **show** False **by** (simp add: sb)
next

case False
show False
proof –

from x-acq **have** $x \in$ all-acquired sb_j
using all-acquired-append [of takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j
dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j]
by auto
moreover
note ownership-distinct [OF i-bound j-bound False ts_{sb}-i jth]
ultimately
show False
using R-owned x-R
by blast
qed
qed
}
thus ?thesis **by** blast
qed

have all-shared-R: $\forall$ i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. i < length ts_{sb} $\longrightarrow$
ts_{sb} ! i = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {}
proof –
{
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j x
assume j-bound: j < length ts_{sb}
assume jth: ts_{sb}!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
assume x-shared: $x \in$ all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)
assume x-R: $x \in R$
have False
proof (cases i=j)
case True **with** x-shared ts_{sb}-i jth **show** False **by** (simp add: sb)
next
case False
show False
proof –
from all-shared-acquired-or-owned [OF sharing-consis [OF j-bound jth]]
have all-shared sb_j ⊆ all-acquired sb_j ∪ $\mathcal{O}_j$.

moreover have all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j) ⊆ all-shared
sb_j
using all-shared-append [of (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)]
by auto
moreover
note ownership-distinct [OF i-bound j-bound False ts_{sb}-i jth]
ultimately
show False
using R-owned x-R x-shared

by blast
qed
qed
}
thus ?thesis **by** blast
qed

from share-all-until-volatile-write-commute [OF ‹ownership-distinct ts_{sb}› ‹sharing-consis $\mathcal{S}_{sb}$ ts_{sb}›
all-shared-L all-shared-A all-acquired-R all-unshared-R all-shared-R]
have share-commute: share-all-until-volatile-write ts_{sb} $\mathcal{S}_{sb} \oplus_W R \ominus_A L$ =
share-all-until-volatile-write ts_{sb} ($\mathcal{S}_{sb} \oplus_W R \ominus_A L$).

{
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j x
assume jth: ts_{sb}!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
assume j-bound: j < length ts_{sb}
assume neq: i $\neq$ j

have release (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
(dom $\mathcal{S}_{sb}$ $\cup$ R $-$ L) $\mathcal{R}_j$
= release (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
(dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$
proof $-$
{
fix a
assume a-in: a $\in$ all-shared (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
have (a $\in$ (dom $\mathcal{S}_{sb}$ $\cup$ R $-$ L)) = (a $\in$ dom $\mathcal{S}_{sb}$)
proof $-$
from A-unowned-by-others [rule-format, OF j-bound neq] jth
A-unacquired-by-others [rule-format, OF - neq] j-bound
have A-dist: A $\cap$ ($\mathcal{O}_j$ $\cup$ all-acquired sb_j) = {}
by (auto dest: all-shared-acquired-in)

from all-shared-acquired-or-owned [OF sharing-consis [OF j-bound jth]] a-in
all-shared-append [of (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
(dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)]
have a-in: a $\in$ $\mathcal{O}_j$ $\cup$ all-acquired sb_j
by auto
with ownership-distinct [OF i-bound j-bound neq ts_{sb}-i jth]
have a $\notin$ ($\mathcal{O}_{sb}$ $\cup$ all-acquired sb) **by** auto

with A-dist R-owned A-R A-shared-owned L-subset a-in
obtain a $\notin$ R **and** a $\notin$ L
by fastforce
then show ?thesis **by** auto
qed
}
then

show ?thesis
apply –
apply (rule release-all-shared-exchange)
apply auto
done
qed
}
note release-commute = this
have (ts_{sb}',m_{sb}(a := f (j_{sb}(t $\mapsto$ m_{sb} a))),$\mathcal{S}_{sb}'$) $\sim$ (ts[i := (p_{sb},is_{sb}',
j_{sb}(t $\mapsto$ ret (m a) (f (j_{sb}(t$\mapsto$m a)))),(), False,$\mathcal{O}_{sb}$ $\cup$ A $-$ R,Map.empty)],m(a := f (j_{sb}(t $\mapsto$ m a))),$\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)
apply (rule sim-config.intros)
apply (simp only: m-a)
apply (simp only: m)
apply (simp only: flush-all-until-volatile-write-update-other [OF a-unflushed$'$, symmetric] ts_{sb}')
apply (simp add: flush-all-until-volatile-nth-update-unused [OF i-bound ts_{sb}-i, simplified sb] sb$'$)
apply (simp add: ts_{sb}' sb$'$ $\mathcal{O}_{sb}'$ m
flush-all-until-volatile-nth-update-unused [OF i-bound ts_{sb}-i, simplified sb])
using share-all-until-volatile-write-RMW-commute [OF i-bound ts_{sb}-i [simplified is_{sb} sb]]
apply (clarsimp simp add: $\mathcal{S}$ ts_{sb}' $\mathcal{S}_{sb}'$ is_{sb} $\mathcal{O}_{sb}'$ $\mathcal{R}_{sb}'$ j_{sb}' sb$'$ sb share-commute)
using leq
apply (simp add: ts_{sb}')
using i-bound i-bound$'$ ts-sim
apply (clarsimp simp add: Let-def nth-list-update
ts_{sb}' sb$'$ sb $\mathcal{O}_{sb}'$ $\mathcal{R}_{sb}'$ $\mathcal{S}_{sb}'$ j_{sb}' $\mathcal{D}_{sb}'$ ex-not m-a
split: if-split-asm)
apply (rule conjI)
apply clarsimp
apply (rule tmps-commute)
apply clarsimp
apply (frule (2) release-commute)
apply clarsimp
apply fastforce
done
ultimately
show ?thesis
using valid-own$'$ valid-hist$'$ valid-reads$'$ valid-sharing$'$ tmps-distinct$'$ valid-sops$'$
valid-dd$'$ load-tmps-fresh$'$ enough-flushs$'$
valid-program-history$'$ valid$'$ m_{sb}' $\mathcal{S}_{sb}'$
by (auto simp del: fun-upd-apply)
next
case (SBHGhost A L R W)
then obtain
is_{sb}: is_{sb} = Ghost A L R W# is_{sb}' **and**
$\mathcal{O}_{sb}'$: $\mathcal{O}_{sb}'$=$\mathcal{O}_{sb}$ **and**
$\mathcal{R}_{sb}'$: $\mathcal{R}_{sb}'$=$\mathcal{R}_{sb}$ **and**
j_{sb}': j_{sb}' = j_{sb} **and**
$\mathcal{D}_{sb}'$: $\mathcal{D}_{sb}'$=$\mathcal{D}_{sb}$ **and**

sb': sb'=sb@[$Ghost_{sb}$ A L R W] **and**
m_{sb}': m_{sb}' = m_{sb} **and**
$\mathcal{S}_{sb}'$: $\mathcal{S}_{sb}'$=$\mathcal{S}_{sb}$
by auto

from safe-memop-flush-sb [simplified is_{sb}] **obtain**
L-subset: L ⊆ A **and**
A-shared-owned: A ⊆ dom (share ?drop-sb $\mathcal{S}$) ∪ acquired True sb $\mathcal{O}_{sb}$ **and**
R-acq: R ⊆ acquired True sb $\mathcal{O}_{sb}$ **and**
A-R: A ∩ R = {} **and**
A-unowned-by-others-ts:
∀j<length (map owned ts). i≠j ⟶ (A ∩ (owned (ts!j) ∪ dom (released (ts!j))) = {})
by cases auto

from A-unowned-by-others-ts ts-sim leq
have A-unowned-by-others:
∀j<length ts_{sb}. i≠j ⟶ (let (-,-,-,sb_j,-,$\mathcal{O}_j$,-) = ts_{sb}!j
in A ∩ (acquired True (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j) $\mathcal{O}_j$ ∪
all-shared (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)) = {})
apply (clarsimp simp add: Let-def)
subgoal for j
apply (drule-tac x=j **in** spec)
apply (force simp add: dom-release-takeWhile)
done
done
have A-unused-by-others:
∀j<length (map $\mathcal{O}$-sb ts_{sb}). i ≠ j ⟶
(let ($\mathcal{O}_j$, sb_j) = map $\mathcal{O}$-sb ts_{sb}! j
in A ∩ outstanding-refs is-volatile-$Write_{sb}$ sb_j = {})
proof –
{
fix j $\mathcal{O}_j$ sb_j
assume j-bound: j < length (map owned ts_{sb})
assume neq-i-j: i≠j
assume ts_{sb}-j: (map $\mathcal{O}$-sb ts_{sb})!j = ($\mathcal{O}_j$,sb_j)
assume conflict: A ∩ outstanding-refs is-volatile-$Write_{sb}$ sb_j ≠ {}
have False
proof –
from j-bound leq
have j-bound′: j < length (map owned ts)
by auto
from j-bound **have** j-bound′′: j < length ts_{sb}
by auto
from j-bound′ **have** j-bound′′′: j < length ts
by simp

from conflict **obtain** a′ **where**
a′-in: a′ ∈ A **and**
a′-in-j: a′ ∈ outstanding-refs is-volatile-$Write_{sb}$ sb_j
by auto

let ?take-sb_j = (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
let ?drop-sb_j = (dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)

from ts-sim [rule-format, OF j-bound$''$] ts_{sb}-j j-bound$''$
obtain p_j $suspends_j$ is_{sbj} j_{sbj} $\mathcal{D}_{sbj}$ $\mathcal{D}_j$ $\mathcal{R}_j$ is_j **where**
ts_{sb}-j: ts_{sb} ! j = (p_j,is_{sbj},j_{sbj}, sb_j,$\mathcal{D}_{sbj}$,$\mathcal{O}_j$,$\mathcal{R}_j$) **and**
$suspends_j$: $suspends_j$ = ?drop-sb_j **and**
$\mathcal{D}_j$: $\mathcal{D}_{sbj}$ = ($\mathcal{D}_j$ $\vee$ outstanding-refs is-volatile-$Write_{sb}$ sb_j $\neq$ {}) **and**
is_j: instrs $suspends_j$ @ is_{sbj} = is_j @ prog-instrs $suspends_j$ **and**
ts_j: ts!j = (hd-prog p_j $suspends_j$, is_j,
j_{sbj} |' (dom j_{sbj} − read-tmps $suspends_j$),(),
$\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$, release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
apply (cases ts_{sb}!j)
apply (force simp add: Let-def)
done

have a$'$ $\in$ outstanding-refs is-volatile-$Write_{sb}$ $suspends_j$
proof −
from a$'$-in-j
have a$'$ $\in$ outstanding-refs is-volatile-$Write_{sb}$ (?take-sb_j @ ?drop-sb_j)
by simp
thus ?thesis
apply (simp only: outstanding-refs-append $suspends_j$)
apply (auto simp add: outstanding-refs-conv dest: set-takeWhileD)
done
qed

from split-volatile-$Write_{sb}$-in-outstanding-refs [OF this]
obtain sop v ys zs A$'$ L$'$ R$'$ W$'$ **where**
split-$suspends_j$: $suspends_j$ = ys @ $Write_{sb}$ True a$'$ sop v A$'$ L$'$ R$'$ W$'$ # zs (**is** $suspends_j$ = ?suspends)
by blast

from direct-memop-step.Ghost [**where** j=j_{sb} **and** m=flush ?drop-sb m]
have (Ghost A L R W# is_{sb}',
j_{sb}, (), flush ?drop-sb m, $\mathcal{D}_{sb}$,
acquired True sb $\mathcal{O}_{sb}$, release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$, share ?drop-sb $\mathcal{S}$) $\rightarrow$
(is_{sb}', j_{sb}, (), flush ?drop-sb m, $\mathcal{D}_{sb}$,
acquired True sb $\mathcal{O}_{sb}$ $\cup$ A − R,
augment-rels (dom (share ?drop-sb $\mathcal{S}$)) R (release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$),
share ?drop-sb $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L).

from direct-computation.concurrent-step.Memop [OF
i-bound-ts$'$ [simplified is_{sb}] ts$'$-i [simplified is_{sb}] this [simplified is_{sb}]]
have store-step: (?ts$'$, flush ?drop-sb m, share ?drop-sb $\mathcal{S}$) $\Rightarrow_d$
(?ts$'$[i := (p_{sb}, is_{sb}', j_{sb}, (),$\mathcal{D}_{sb}$, acquired True sb $\mathcal{O}_{sb}$ $\cup$ A − R,augment-rels (dom (share ?drop-sb $\mathcal{S}$)) R (release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$))],
flush ?drop-sb m,share ?drop-sb $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)
(**is** - $\Rightarrow_d$ (?ts-A, ?m-A, ?share-A))

by (simp add: is_{sb})

from i-bound′ **have** i-bound″: i < length ?ts-A
by simp

from valid-program-history [OF j-bound″ ts_{sb}-j]
have causal-program-history is_{sbj} sb_j.
then have cph: causal-program-history is_{sbj} ?suspends
apply −
apply (rule causal-program-history-suffix [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp add: split-$suspends_j$)
done

from ts_j neq-i-j j-bound
have ts-A-j: ?ts-A!j = (hd-prog p_j (ys @ $Write_{sb}$ True a′ sop v A′ L′ R′ W′ # zs), is_j, j_{sbj} |‘ (dom j_{sbj} − read-tmps (ys @ $Write_{sb}$ True a′ sop v A′ L′ R′ W′ # zs)), (), $\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$,release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
by (simp add: split-$suspends_j$)

from j-bound‴ i-bound′ neq-i-j **have** j-bound⁗: j < length ?ts-A
by simp

from valid-last-prog [OF j-bound″ ts_{sb}-j] **have** last-prog: last-prog p_j sb_j = p_j.
then
have lp: last-prog p_j ?suspends = p_j
apply −
apply (rule last-prog-same-append [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

from valid-reads [OF j-bound″ ts_{sb}-j]
have reads-consis: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.

from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts_{sb}› j-bound″
ts_{sb}-j reads-consis]
have reads-consis-m: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
by (simp add: m $suspends_j$)

from outstanding-non-write-non-vol-reads-drop-disj [OF i-bound j-bound″ neq-i-j ts_{sb}-i ts_{sb}-j]
have outstanding-refs is-$Write_{sb}$?drop-sb ∩ outstanding-refs is-non-volatile-$Read_{sb}$ $suspends_j$ = {}
by (simp add: $suspends_j$)
from reads-consistent-flush-independent [OF this reads-consis-m]
have reads-consis-flush-m: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$)

?m-A $suspends_j$.
hence reads-consis-m-A-ys: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) ?m-A ys
by (simp add: split-$suspends_j$ reads-consistent-append)

from valid-history [OF j-bound″ ts_{sb}-j]
have h-consis:
history-consistent j_{sbj} (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof −
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from valid-write-sops [OF j-bound″ ts_{sb}-j]
have ∀sop∈write-sops (?take-sb_j@?suspends). valid-sop sop
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain valid-sops-take: ∀sop∈write-sops ?take-sb_j. valid-sop sop **and**
valid-sops-drop: ∀sop∈write-sops ys. valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound″ ts_{sb}-j]
have distinct-read-tmps (?take-sb_j@$suspends_j$)
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb_j ∩ read-tmps $suspends_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps $suspends_j$
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp only: distinct-read-tmps-append)
done

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop h-consis]
last-prog-hd-prog

have hist-consis′: history-consistent j_{sbj} (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis]
have no-vol-read: outstanding-refs is-volatile-$Read_{sb}$ ys = {}
by (auto simp add: outstanding-refs-append $suspends_j$ [symmetric]
split-$suspends_j$)

from flush-store-buffer-append [
OF j-bound′′′′ is_j [simplified split-$suspends_j$] cph [simplified $suspends_j$]
ts-A-j [simplified split-$suspends_j$] refl lp [simplified split-$suspends_j$] reads-consis-m-A-ys
hist-consis′ [simplified split-$suspends_j$] valid-sops-drop distinct-read-tmps-drop
[simplified split-$suspends_j$]
no-volatile-$Read_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=?share-A]
obtain is_j′ $\mathcal{R}_j$′ **where**
is_j′: instrs ($Write_{sb}$ True a′ sop v A′ L′ R′ W′ # zs) @ is_{sbj} =
is_j′ @ prog-instrs ($Write_{sb}$ True a′ sop v A′ L′ R′ W′ # zs) **and**
steps-ys: (?ts-A, ?m-A, ?share-A) $\Rightarrow_d{}^*$
(?ts-A[j:= (last-prog (hd-prog p_j ($Write_{sb}$ True a′ sop v A′ L′ R′ W′ # zs)) ys,
is_j′,
j_{sbj} |‘ (dom j_{sbj} − read-tmps ($Write_{sb}$ True a′ sop v A′ L′ R′ W′ #
zs)),(),
$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ ys ≠ {}, acquired True ys
(acquired True ?take-sb_j $\mathcal{O}_j$),$\mathcal{R}_j$′)],
flush ys ?m-A,
share ys ?share-A)
(**is** (-,-,-) $\Rightarrow_d{}^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto)

note conflict-computation = rtranclp-trans [OF rtranclp-r-rtranclp [OF steps-flush-sb,
OF store-step] steps-ys]
from cph
have causal-program-history is_{sbj} ((ys @ [$Write_{sb}$ True a′ sop v A′ L′ R′ W′]) @ zs)
by simp
from causal-program-history-suffix [OF this]
have cph′: causal-program-history is_{sbj} zs.
interpret $causal_j$: causal-program-history is_{sbj} zs **by** (rule cph′)

from $causal_j$.causal-program-history [of [], simplified, OF refl] is_j′
obtain is_j′′
where is_j′: is_j′ = Write True a′ sop A′ L′ R′ W′ #is_j′′ **and**
is_j′′: instrs zs @ is_{sbj} = is_j′′ @ prog-instrs zs
by clarsimp

from j-bound′′′
have j-bound-ys: j < length ?ts-ys
by auto

from j-bound-ys neq-i-j

have ts-ys-j: ?ts-ys!j=(last-prog (hd-prog p_j (Write$_{sb}$ True a′ sop v A′ L′ R′ W′# zs))
ys, is_j′,
j_{sbj} |‘ (dom j_{sbj} − read-tmps (Write$_{sb}$ True a′ sop v A′ L′ R′ W′# zs)),(),
$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-Write$_{sb}$ ys ≠ {},
acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$),$\mathcal{R}_j$′)
by auto

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys).

from safe-delayedE [OF this j-bound-ys ts-ys-j, simplified is_j′]
have a-unowned:
∀ i < length ?ts-ys. j≠i ⟶ (let ($\mathcal{O}_i$) = map owned ?ts-ys!i in a′ ∉ $\mathcal{O}_i$)
apply cases
apply (auto simp add: Let-def is_{sb})
done
from a′-in a-unowned [rule-format, of i] neq-i-j i-bound′ A-R
show False
by (auto simp add: Let-def)
qed
}
thus ?thesis
by (auto simp add: Let-def)
qed

have A-unaquired-by-others:
∀j<length (map $\mathcal{O}$-sb ts_{sb}). i ≠ j ⟶
(let ($\mathcal{O}_j$, sb_j) = map $\mathcal{O}$-sb ts_{sb}! j
in A ∩ all-acquired sb_j = {})
proof −
{
fix j $\mathcal{O}_j$ sb_j
assume j-bound: j < length (map owned ts_{sb})
assume neq-i-j: i≠j
assume ts_{sb}-j: (map $\mathcal{O}$-sb ts_{sb})!j = ($\mathcal{O}_j$,sb_j)
assume conflict: A ∩ all-acquired sb_j ≠ {}
have False
proof −
from j-bound leq
have j-bound′: j < length (map owned ts)
by auto
from j-bound **have** j-bound″: j < length ts_{sb}
by auto
from j-bound′ **have** j-bound‴: j < length ts
by simp

from conflict **obtain** a′ **where**
a′-in: a′ ∈ A **and**
a′-in-j: a′ ∈ all-acquired sb_j
by auto

let ?take-sb_j = (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)
let ?drop-sb_j = (dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j)

from ts-sim [rule-format, OF j-bound″] ts_{sb}-j j-bound″
obtain p_j $suspends_j$ is_{sbj} j_{sbj} $\mathcal{D}_{sbj}$ $\mathcal{D}_j$ $\mathcal{R}_j$ is_j **where**
ts_{sb}-j: ts_{sb} ! j = (p_j,is_{sbj}, j_{sbj}, sb_j,$\mathcal{D}_{sbj}$,$\mathcal{O}_j$,$\mathcal{R}_j$) **and**
$suspends_j$: $suspends_j$ = ?drop-sb_j **and**
$\mathcal{D}_j$: $\mathcal{D}_{sbj}$ = ($\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ sb_j ≠ {}) **and**
is_j: instrs $suspends_j$ @ is_{sbj} = is_j @ prog-instrs $suspends_j$ **and**
ts_j: ts!j = (hd-prog p_j $suspends_j$, is_j,
j_{sbj} |‘ (dom j_{sbj} − read-tmps $suspends_j$),(),
$\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$,release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
apply (cases ts_{sb}!j)
apply (force simp add: Let-def)
done

from a′-in-j all-acquired-append [of ?take-sb_j ?drop-sb_j]
have a′ ∈ all-acquired ?take-sb_j ∨ a′ ∈ all-acquired $suspends_j$
by (auto simp add: $suspends_j$)
thus False
proof
assume a′ ∈ all-acquired ?take-sb_j
with A-unowned-by-others [rule-format, OF - neq-i-j] ts_{sb}-j j-bound a′-in
show False
by (auto dest: all-acquired-unshared-acquired)
next
assume conflict-drop: a′ ∈ all-acquired $suspends_j$
from split-all-acquired-in [OF conflict-drop]

show False
proof
assume ∃sop a″ v ys zs A L R W.
$suspends_j$ = ys @ $Write_{sb}$ True a″ sop v A L R W# zs ∧ a′ ∈ A
then
obtain a″ sop′ v′ ys zs A′ L′ R′ W′ **where**
split-$suspends_j$: $suspends_j$ = ys @ $Write_{sb}$ True a″ sop′ v′ A′ L′ R′ W′ # zs
(**is** $suspends_j$ = ?suspends) **and**
a′-A′: a′ ∈ A′
by auto

from direct-memop-step.Ghost [**where** j=j_{sb} **and** m=flush ?drop-sb m]
have (Ghost A L R W# is_{sb}′,
j_{sb}, (), flush ?drop-sb m,$\mathcal{D}_{sb}$,
acquired True sb $\mathcal{O}_{sb}$, release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$,share ?drop-sb $\mathcal{S}$) →
(is_{sb}′, j_{sb}, (), flush ?drop-sb m, $\mathcal{D}_{sb}$,
acquired True sb $\mathcal{O}_{sb}$ ∪ A − R,
augment-rels (dom (share ?drop-sb $\mathcal{S}$)) R (release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$),
share ?drop-sb $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L).

from direct-computation.concurrent-step.Memop [OF
i-bound-ts′ [simplified is_{sb}] ts′-i [simplified is_{sb}] this [simplified is_{sb}]]
have store-step: (?ts′, flush ?drop-sb m, share ?drop-sb $\mathcal{S}$) $\Rightarrow_d$
(?ts′[i := (p_{sb}, is_{sb}′,j_{sb}, (),$\mathcal{D}_{sb}$,
acquired True sb $\mathcal{O}_{sb}$ ∪ A − R,
augment-rels (dom (share ?drop-sb $\mathcal{S}$)) R (release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$))],

flush ?drop-sb m,share ?drop-sb $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)
(**is** - $\Rightarrow_d$ (?ts-A, ?m-A, ?share-A))
by (simp add: is_{sb})

from i-bound′ **have** i-bound″: i < length ?ts-A
by simp

from valid-program-history [OF j-bound″ ts_{sb}-j]
have causal-program-history is_{sbj} sb_j.
then have cph: causal-program-history is_{sbj} ?suspends
apply −
apply (rule causal-program-history-suffix [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp add: split-$suspends_j$)
done

from ts_j neq-i-j j-bound
have ts-A-j: ?ts-A!j = (hd-prog p_j (ys @ $Write_{sb}$ True a″ sop′ v′ A′ L′ R′ W′ # zs),
is_j,
j_{sbj} |‘ (dom j_{sbj} − read-tmps (ys @ $Write_{sb}$ True a″ sop′ v′ A′ L′ R′ W′ # zs)), (), $\mathcal{D}_j$,
acquired True ?take-sb_j $\mathcal{O}_j$, release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
by (simp add: split-$suspends_j$)

from j-bound‴ i-bound′ neq-i-j **have** j-bound⁗: j < length ?ts-A
by simp

from valid-last-prog [OF j-bound″ ts_{sb}-j] **have** last-prog: last-prog p_j sb_j = p_j.
then
have lp: last-prog p_j ?suspends = p_j
apply −
apply (rule last-prog-same-append [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

from valid-reads [OF j-bound″ ts_{sb}-j]
have reads-consis: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.

from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing
$\mathcal{S}_{sb}$ ts_{sb}› j-bound″

ts_{sb}-j reads-consis]
have reads-consis-m: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
by (simp add: m $suspends_j$)

from outstanding-non-write-non-vol-reads-drop-disj [OF i-bound j-bound″ neq-i-j ts_{sb}-i ts_{sb}-j]
have outstanding-refs is-$Write_{sb}$?drop-sb ∩ outstanding-refs is-non-volatile-$Read_{sb}$ $suspends_j$ = {}
by (simp add: $suspends_j$)
from reads-consistent-flush-independent [OF this reads-consis-m]
have reads-consis-flush-m: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) ?m-A $suspends_j$.
hence reads-consis-m-A-ys: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) ?m-A ys
by (simp add: split-$suspends_j$ reads-consistent-append)

from valid-history [OF j-bound″ ts_{sb}-j]
have h-consis:
history-consistent j_{sbj} (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof −
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from valid-write-sops [OF j-bound″ ts_{sb}-j]
have ∀sop∈write-sops (?take-sb_j@?suspends). valid-sop sop
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain valid-sops-take: ∀sop∈write-sops ?take-sb_j. valid-sop sop **and**
valid-sops-drop: ∀sop∈write-sops ys. valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound″ ts_{sb}-j]
have distinct-read-tmps (?take-sb_j@$suspends_j$)

by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb$_j$ $\cap$ read-tmps suspends$_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps suspends$_j$
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
apply (simp only: distinct-read-tmps-append)
done

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop h-consis]

last-prog-hd-prog
have hist-consis$'$: history-consistent j_{sbj} (hd-prog p_j suspends$_j$) suspends$_j$
by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis]
have no-vol-read: outstanding-refs is-volatile-Read$_{sb}$ ys = {}
by (auto simp add: outstanding-refs-append suspends$_j$ [symmetric]
split-suspends$_j$)

from flush-store-buffer-append [
OF j-bound$''''$ is$_j$ [simplified split-suspends$_j$] cph [simplified suspends$_j$]
ts-A-j [simplified split-suspends$_j$] refl lp [simplified split-suspends$_j$] reads-consis-m-A-ys
hist-consis$'$ [simplified split-suspends$_j$] valid-sops-drop distinct-read-tmps-drop
[simplified split-suspends$_j$]
no-volatile-Read$_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=?share-A]
obtain is$_j$$'$ $\mathcal{R}_j$$'$ **where**
is$_j$$'$: instrs (Write$_{sb}$ True a$''$ sop$'$ v$'$ A$'$ L$'$ R$'$ W$'$ # zs) @ is$_{sbj}$ =
is$_j$$'$ @ prog-instrs (Write$_{sb}$ True a$''$ sop$'$ v$'$ A$'$ L$'$ R$'$ W$'$ # zs) **and**
steps-ys: (?ts-A, ?m-A, ?share-A) $\Rightarrow_d{}^*$
(?ts-A[j:= (last-prog (hd-prog p_j (Write$_{sb}$ True a$''$ sop$'$ v$'$ A$'$ L$'$ R$'$ W$'$ # zs)) ys,
is$_j$$'$,
j_{sbj} |‘ (dom j_{sbj} − read-tmps (Write$_{sb}$ True a$''$ sop$'$ v$'$ A$'$ L$'$ R$'$ W$'$ # zs)),(),
$\mathcal{D}_j$ $\vee$ outstanding-refs is-volatile-Write$_{sb}$ ys $\neq$ {}, acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$),$\mathcal{R}_j$$'$)],
flush ys ?m-A,share ys ?share-A)
(**is** (-,-,-) $\Rightarrow_d{}^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto)

note conflict-computation = rtranclp-trans [OF rtranclp-r-rtranclp [OF steps-flush-sb, OF store-step] steps-ys]
from cph
have causal-program-history is$_{sbj}$ ((ys @ [Write$_{sb}$ True a$''$ sop$'$ v$'$ A$'$ L$'$ R$'$ W$'$]) @ zs)
by simp
from causal-program-history-suffix [OF this]
have cph$'$: causal-program-history is$_{sbj}$ zs**.**
interpret causal$_j$: causal-program-history is$_{sbj}$ zs **by** (rule cph$'$)

from causal$_j$.causal-program-history [of [], simplified, OF refl] is$_j$′
obtain is$_j$″
where is$_j$′: is$_j$′ = Write True a″ sop′ A′ L′ R′ W′#is$_j$″ **and**
is$_j$″: instrs zs @ is$_{sbj}$ = is$_j$″ @ prog-instrs zs
by clarsimp

from j-bound‴
have j-bound-ys: j < length ?ts-ys
by auto

from j-bound-ys neq-i-j
have ts-ys-j: ?ts-ys!j=(last-prog (hd-prog p$_j$ (Write$_{sb}$ True a″ sop′ v′ A′ L′ R′ W′# zs)) ys, is$_j$′,
j$_{sbj}$ |‘ (dom j$_{sbj}$ − read-tmps (Write$_{sb}$ True a″ sop′ v′ A′ L′ R′ W′# zs)),(),
$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-Write$_{sb}$ ys ≠ {},
acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$),$\mathcal{R}_j$′)
by auto

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys).

from safe-delayedE [OF this j-bound-ys ts-ys-j, simplified is$_j$′]
have A′-unowned:
∀i < length ?ts-ys. j≠i ⟶ (let ($\mathcal{O}_i$) = map owned ?ts-ys!i in A′ ∩ $\mathcal{O}_i$ = {})
apply cases
apply (fastforce simp add: Let-def is$_{sb}$)+
done
from a′-in a′-A′ A′-unowned [rule-format, of i] neq-i-j i-bound′ A-R
show False
by (auto simp add: Let-def)
next
assume ∃A L R W ys zs.
suspends$_j$ = ys @ Ghost$_{sb}$ A L R W # zs ∧ a′ ∈ A
then
obtain ys zs A′ L′ R′ W′ **where**
split-suspends$_j$: suspends$_j$ = ys @ Ghost$_{sb}$ A′ L′ R′ W′# zs (**is** suspends$_j$ = ?suspends)
and
a′-A′: a′ ∈ A′
by auto

from direct-memop-step.Ghost [**where** j=j$_{sb}$ **and** m=flush ?drop-sb m]
have (Ghost A L R W# is$_{sb}$′,
j$_{sb}$, (), flush ?drop-sb m, $\mathcal{D}_{sb}$,
acquired True sb $\mathcal{O}_{sb}$, release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$, share ?drop-sb $\mathcal{S}$) →
(is$_{sb}$′, j$_{sb}$, (), flush ?drop-sb m, $\mathcal{D}_{sb}$,
acquired True sb $\mathcal{O}_{sb}$ ∪ A − R,
augment-rels (dom (share ?drop-sb $\mathcal{S}$)) R (release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$),
share ?drop-sb $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L).

from direct-computation.concurrent-step.Memop [OF

i-bound-ts′ [simplified is_{sb}] ts′-i [simplified is_{sb}] this [simplified is_{sb}]]
have store-step: (?ts′, flush ?drop-sb m, share ?drop-sb $\mathcal{S}$) $\Rightarrow_d$
(?ts′[i := (p_{sb}, is_{sb}′, j_{sb}, (), $\mathcal{D}_{sb}$, acquired True sb $\mathcal{O}_{sb}$ ∪ A − R,augment-rels (dom (share ?drop-sb $\mathcal{S}$)) R (release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$))],
flush ?drop-sb m,share ?drop-sb $\mathcal{S} \oplus_W$ R $\ominus_A$ L)
(**is** - $\Rightarrow_d$ (?ts-A, ?m-A, ?share-A))
by (simp add: is_{sb})

from i-bound′ **have** i-bound″: i < length ?ts-A
by simp

from valid-program-history [OF j-bound″ ts_{sb}-j]
have causal-program-history is_{sbj} sb_j.
then have cph: causal-program-history is_{sbj} ?suspends
apply −
apply (rule causal-program-history-suffix [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp add: split-$suspends_j$)
done

from ts_j neq-i-j j-bound
have ts-A-j: ?ts-A!j = (hd-prog p_j (ys @ $Ghost_{sb}$ A′ L′ R′ W′# zs), is_j,
j_{sbj} |‘ (dom j_{sbj} − read-tmps (ys @ $Ghost_{sb}$ A′ L′ R′ W′# zs)), (),$\mathcal{D}_j$,
acquired True ?take-sb_j $\mathcal{O}_j$,release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
by (simp add: split-$suspends_j$)

from j-bound‴ i-bound′ neq-i-j **have** j-bound⁗: j < length ?ts-A
by simp

from valid-last-prog [OF j-bound″ ts_{sb}-j] **have** last-prog: last-prog p_j sb_j = p_j.
then
have lp: last-prog p_j ?suspends = p_j
apply −
apply (rule last-prog-same-append [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done
from valid-reads [OF j-bound″ ts_{sb}-j]
have reads-consis: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.

from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts_{sb}› j-bound″
ts_{sb}-j reads-consis]
have reads-consis-m: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
by (simp add: m $suspends_j$)

from outstanding-non-write-non-vol-reads-drop-disj [OF i-bound j-bound″ neq-i-j ts_{sb}-i ts_{sb}-j]

have outstanding-refs is-Write$_{sb}$?drop-sb ∩ outstanding-refs is-non-volatile-Read$_{sb}$ suspends$_j$ = {}
by (simp add: suspends$_j$)
from reads-consistent-flush-independent [OF this reads-consis-m]
have reads-consis-flush-m: reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?m-A suspends$_j$.
hence reads-consis-m-A-ys: reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?m-A ys
by (simp add: split-suspends$_j$ reads-consistent-append)

from valid-history [OF j-bound$''$ ts$_{sb}$-j]
have h-consis:
history-consistent j$_{sbj}$ (hd-prog p$_j$ (?take-sb$_j$@suspends$_j$)) (?take-sb$_j$@suspends$_j$)
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p$_j$ sb$_j$) ?take-sb$_j$ = (hd-prog p$_j$ suspends$_j$)
proof −
from last-prog **have** last-prog p$_j$ (?take-sb$_j$@?drop-sb$_j$) = p$_j$
by simp
from last-prog-hd-prog-append$'$ [OF h-consis] this
have last-prog (hd-prog p$_j$ suspends$_j$) ?take-sb$_j$ = hd-prog p$_j$ suspends$_j$
by (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
moreover
have last-prog (hd-prog p$_j$ (?take-sb$_j$ @ suspends$_j$)) ?take-sb$_j$ =
last-prog (hd-prog p$_j$ suspends$_j$) ?take-sb$_j$
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
qed

from valid-write-sops [OF j-bound$''$ ts$_{sb}$-j]
have ∀ sop∈write-sops (?take-sb$_j$@?suspends). valid-sop sop
by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
then obtain valid-sops-take: ∀ sop∈write-sops ?take-sb$_j$. valid-sop sop **and**
valid-sops-drop: ∀ sop∈write-sops ys. valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound$''$ ts$_{sb}$-j]
have distinct-read-tmps (?take-sb$_j$@suspends$_j$)
by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb$_j$ ∩ read-tmps suspends$_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps suspends$_j$
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)

apply (simp only: distinct-read-tmps-append)
done

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop h-consis]

last-prog-hd-prog
have hist-consis′: history-consistent j_{sbj} (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis]
have no-vol-read: outstanding-refs is-volatile-$Read_{sb}$ ys = {}
by (auto simp add: outstanding-refs-append $suspends_j$ [symmetric]
split-$suspends_j$)

from flush-store-buffer-append [
OF j-bound′′′′ is_j [simplified split-$suspends_j$] cph [simplified $suspends_j$]
ts-A-j [simplified split-$suspends_j$] refl lp [simplified split-$suspends_j$] reads-consis-m-A-ys
hist-consis′ [simplified split-$suspends_j$] valid-sops-drop distinct-read-tmps-drop [simplified split-$suspends_j$]
no-volatile-$Read_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=?share-A]
obtain is_j′ $\mathcal{R}_j$′ **where**
is_j′: instrs ($Ghost_{sb}$ A′ L′ R′ W′# zs) @ is_{sbj} =
is_j′ @ prog-instrs ($Ghost_{sb}$ A′ L′ R′ W′# zs) **and**
steps-ys: (?ts-A, ?m-A, ?share-A) $\Rightarrow_d{}^*$
(?ts-A[j:= (last-prog (hd-prog p_j ($Ghost_{sb}$ A′ L′ R′ W′# zs)) ys,
is_j′,
j_{sbj} |‘ (dom j_{sbj} − read-tmps ($Ghost_{sb}$ A′ L′ R′ W′# zs)),(),
$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ ys ≠ {}, acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$),$\mathcal{R}_j$′)],
flush ys ?m-A, share ys ?share-A)
(**is** (-,-,-) $\Rightarrow_d{}^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto)

note conflict-computation = rtranclp-trans [OF rtranclp-r-rtranclp [OF steps-flush-sb, OF store-step] steps-ys]
from cph
have causal-program-history is_{sbj} ((ys @ [$Ghost_{sb}$ A′ L′ R′ W′]) @ zs)
by simp
from causal-program-history-suffix [OF this]
have cph′: causal-program-history is_{sbj} zs**.**
interpret $causal_j$: causal-program-history is_{sbj} zs **by** (rule cph′)

from $causal_j$.causal-program-history [of [], simplified, OF refl] is_j′
obtain is_j′′
where is_j′: is_j′ = Ghost A′ L′ R′ W′#is_j′′ **and**
is_j′′: instrs zs @ is_{sbj} = is_j′′ @ prog-instrs zs
by clarsimp

from j-bound′′′

have j-bound-ys: j < length ?ts-ys
by auto

from j-bound-ys neq-i-j
have ts-ys-j: ?ts-ys!j=(last-prog (hd-prog p_j (Ghost$_{sb}$ A′ L′ R′ W′# zs)) ys, is_j′,
j_{sbj} |‘ (dom j_{sbj} − read-tmps (Write$_{sb}$ True a″ sop′ v′ A′ L′ R′ W′# zs)),(),
$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-Write$_{sb}$ ys ≠ {},
acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$),$\mathcal{R}_j$′)
by auto

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys).

from safe-delayedE [OF this j-bound-ys ts-ys-j, simplified is_j′]
have A′-unowned:
∀i < length ?ts-ys. j≠i ⟶ (let ($\mathcal{O}_i$) = map owned ?ts-ys!i in A′ ∩ $\mathcal{O}_i$ = {})
apply cases
apply (fastforce simp add: Let-def is_{sb})+
done
from a′-in a′-A′ A′-unowned [rule-format, of i] neq-i-j i-bound′ A-R
show False
by (auto simp add: Let-def)
qed
qed
qed
}
thus ?thesis
by (auto simp add: Let-def)
qed

have A-no-read-only-reads-by-others:
∀j<length (map $\mathcal{O}$-sb ts_{sb}). i ≠ j ⟶
(let ($\mathcal{O}_j$, sb_j) = map $\mathcal{O}$-sb ts_{sb}! j
in A ∩ read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)
$\mathcal{O}_j$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j) = {})
proof −
{
fix j $\mathcal{O}_j$ sb_j
assume j-bound: j < length (map owned ts_{sb})
assume neq-i-j: i≠j
assume ts_{sb}-j: (map $\mathcal{O}$-sb ts_{sb})!j = ($\mathcal{O}_j$,sb_j)
let ?take-sb_j = (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)
let ?drop-sb_j = (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)

assume conflict: A ∩ read-only-reads (acquired True ?take-sb_j $\mathcal{O}_j$) ?drop-sb_j ≠ {}
have False
proof −
from j-bound leq
have j-bound′: j < length (map owned ts)

by auto
from j-bound **have** j-bound$''$: j $<$ length ts_{sb}
by auto
from j-bound$'$ **have** j-bound$'''$: j $<$ length ts
by simp

from conflict **obtain** a$'$ **where**
a$'$-in: a$'$ $\in$ A **and**
a$'$-in-j: a$'$ $\in$ read-only-reads (acquired True ?take-sb_j $\mathcal{O}_j$) ?drop-sb_j
by auto

from ts-sim [rule-format, OF j-bound$''$] ts_{sb}-j j-bound$''$
obtain p_j $suspends_j$ is_{sbj} $\mathcal{D}_{sbj}$ $\mathcal{D}_j$ $\mathcal{R}_j$ j_{sbj} is_j **where**
ts_{sb}-j: ts_{sb} ! j = (p_j,is_{sbj}, j_{sbj}, sb_j,$\mathcal{D}_{sbj}$,$\mathcal{O}_j$,$\mathcal{R}_j$) **and**
$suspends_j$: $suspends_j$ = ?drop-sb_j **and**
is_j: instrs $suspends_j$ @ is_{sbj} = is_j @ prog-instrs $suspends_j$ **and**
$\mathcal{D}_j$: $\mathcal{D}_{sbj}$ = ($\mathcal{D}_j$ $\vee$ outstanding-refs is-volatile-$Write_{sb}$ sb_j $\neq$ {}) **and**
ts_j: ts!j = (hd-prog p_j $suspends_j$, is_j,
j_{sbj} |' (dom j_{sbj} $-$ read-tmps $suspends_j$),(), $\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$,release ?take-sb_j (dom $\mathcal{S}_{sb}$) $\mathcal{R}_j$)
apply (cases ts_{sb}!j)
apply (force simp add: Let-def)
done

from split-in-read-only-reads [OF a$'$-in-j [simplified $suspends_j$ [symmetric]]]
obtain t v ys zs **where**
split-$suspends_j$: $suspends_j$ = ys @ $Read_{sb}$ False a$'$ t v# zs (**is** $suspends_j$ = ?suspends) **and**
a$'$-unacq: a$'$ $\notin$ acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$)
by blast

from direct-memop-step.Ghost [**where** j=j_{sb} **and** m=flush ?drop-sb m]
have (Ghost A L R W# $is_{sb}$$'$,
j_{sb}, (), flush ?drop-sb m, $\mathcal{D}_{sb}$,
acquired True sb $\mathcal{O}_{sb}$, release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$, share ?drop-sb $\mathcal{S}$) $\rightarrow$
($is_{sb}$$'$, j_{sb}, (), flush ?drop-sb m, $\mathcal{D}_{sb}$,
acquired True sb $\mathcal{O}_{sb}$ $\cup$ A $-$ R,
augment-rels (dom (share ?drop-sb $\mathcal{S}$)) R (release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$),
share ?drop-sb $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)**.**

from direct-computation.concurrent-step.Memop [OF
i-bound-ts$'$ [simplified is_{sb}] ts$'$-i [simplified is_{sb}] this [simplified is_{sb}]]
have store-step: (?ts$'$, flush ?drop-sb m, share ?drop-sb $\mathcal{S}$) $\Rightarrow_d$
(?ts$'$[i := (p_{sb}, $is_{sb}$$'$, j_{sb}, (),$\mathcal{D}_{sb}$, acquired True sb $\mathcal{O}_{sb}$ $\cup$ A $-$ R,augment-rels (dom (share ?drop-sb $\mathcal{S}$)) R (release sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$))],
flush ?drop-sb m,share ?drop-sb $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)
(**is** - $\Rightarrow_d$ (?ts-A, ?m-A, ?share-A))

by (simp add: is_{sb})

from i-bound′ **have** i-bound″: i < length ?ts-A
by simp

from valid-program-history [OF j-bound″ ts_{sb}-j]
have causal-program-history is_{sbj} sb_{j}.
then have cph: causal-program-history is_{sbj} ?suspends
apply −
apply (rule causal-program-history-suffix [**where** sb=?take-sb_{j}])
apply (simp only: split-$\text{suspends}_{\mathsf{j}}$ [symmetric] $\text{suspends}_{\mathsf{j}}$)
apply (simp add: split-$\text{suspends}_{\mathsf{j}}$)
done

from ts_{j} neq-i-j j-bound
have ts-A-j: ?ts-A!j = (hd-prog p_{j} (ys @ $\text{Read}_{\mathsf{sb}}$ False a′ t v# zs), is_{j},
j_{sbj} |‘ (dom j_{sbj} − read-tmps (ys @ $\text{Read}_{\mathsf{sb}}$ False a′ t v# zs)), (),$\mathcal{D}_{\mathsf{j}}$,
acquired True ?take-sb_{j} $\mathcal{O}_{\mathsf{j}}$,release ?take-sb_{j} (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}_{\mathsf{j}}$)
by (simp add: split-$\text{suspends}_{\mathsf{j}}$)

from j-bound‴ i-bound′ neq-i-j **have** j-bound⁗: j < length ?ts-A
by simp

from valid-last-prog [OF j-bound″ ts_{sb}-j] **have** last-prog: last-prog p_{j} sb_{j} = p_{j}.
then
have lp: last-prog p_{j} ?suspends = p_{j}
apply −
apply (rule last-prog-same-append [**where** sb=?take-sb_{j}])
apply (simp only: split-$\text{suspends}_{\mathsf{j}}$ [symmetric] $\text{suspends}_{\mathsf{j}}$)
apply simp
done
from valid-reads [OF j-bound″ ts_{sb}-j]
have reads-consis: reads-consistent False $\mathcal{O}_{\mathsf{j}}$ m_{sb} sb_{j}.

from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{\mathsf{sb}}$ ts_{sb}› j-bound″
ts_{sb}-j reads-consis]
have reads-consis-m: reads-consistent True (acquired True ?take-sb_{j} $\mathcal{O}_{\mathsf{j}}$) m $\text{suspends}_{\mathsf{j}}$
by (simp add: m $\text{suspends}_{\mathsf{j}}$)

from outstanding-non-write-non-vol-reads-drop-disj [OF i-bound j-bound″ neq-i-j ts_{sb}-i ts_{sb}-j]
have outstanding-refs is-$\text{Write}_{\mathsf{sb}}$?drop-sb ∩ outstanding-refs is-non-volatile-$\text{Read}_{\mathsf{sb}}$ $\text{suspends}_{\mathsf{j}}$ = {}
by (simp add: $\text{suspends}_{\mathsf{j}}$)
from reads-consistent-flush-independent [OF this reads-consis-m]
have reads-consis-flush-m: reads-consistent True (acquired True ?take-sb_{j} $\mathcal{O}_{\mathsf{j}}$)
?m-A $\text{suspends}_{\mathsf{j}}$.

hence reads-consis-m-A-ys: reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ?m-A ys
by (simp add: split-suspends$_j$ reads-consistent-append)

from valid-history [OF j-bound$''$ ts$_{sb}$-j]
have h-consis:
history-consistent j$_{sbj}$ (hd-prog p$_j$ (?take-sb$_j$@suspends$_j$)) (?take-sb$_j$@suspends$_j$)
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p$_j$ sb$_j$) ?take-sb$_j$ = (hd-prog p$_j$ suspends$_j$)
proof –
from last-prog **have** last-prog p$_j$ (?take-sb$_j$@?drop-sb$_j$) = p$_j$
by simp
from last-prog-hd-prog-append$'$ [OF h-consis] this
have last-prog (hd-prog p$_j$ suspends$_j$) ?take-sb$_j$ = hd-prog p$_j$ suspends$_j$
by (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
moreover
have last-prog (hd-prog p$_j$ (?take-sb$_j$ @ suspends$_j$)) ?take-sb$_j$ =
last-prog (hd-prog p$_j$ suspends$_j$) ?take-sb$_j$
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
qed

from valid-write-sops [OF j-bound$''$ ts$_{sb}$-j]
have $\forall$sop$\in$write-sops (?take-sb$_j$@?suspends). valid-sop sop
by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
then obtain valid-sops-take: $\forall$sop$\in$write-sops ?take-sb$_j$. valid-sop sop **and**
valid-sops-drop: $\forall$sop$\in$write-sops ys. valid-sop sop
apply (simp only: write-sops-append)
apply auto
done

from read-tmps-distinct [OF j-bound$''$ ts$_{sb}$-j]
have distinct-read-tmps (?take-sb$_j$@suspends$_j$)
by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb$_j$ $\cap$ read-tmps suspends$_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps suspends$_j$
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
apply (simp only: distinct-read-tmps-append)
done

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop h-consis]
last-prog-hd-prog
have hist-consis$'$: history-consistent j$_{sbj}$ (hd-prog p$_j$ suspends$_j$) suspends$_j$

by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis]
have no-vol-read: outstanding-refs is-volatile-Read$_{sb}$ ys = {}
by (auto simp add: outstanding-refs-append suspends$_j$ [symmetric]
split-suspends$_j$)

from flush-store-buffer-append [
OF j-bound'''' is$_j$ [simplified split-suspends$_j$] cph [simplified suspends$_j$]
ts-A-j [simplified split-suspends$_j$] refl lp [simplified split-suspends$_j$] reads-consis-m-A-ys
hist-consis' [simplified split-suspends$_j$] valid-sops-drop distinct-read-tmps-drop
[simplified split-suspends$_j$]
no-volatile-Read$_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=?share-A]
obtain is$_j$' $\mathcal{R}_j$' **where**
is$_j$': instrs (Read$_{sb}$ False a' t v # zs) @ is$_{sbj}$ =
is$_j$' @ prog-instrs (Read$_{sb}$ False a' t v # zs) **and**
steps-ys: (?ts-A, ?m-A, ?share-A) $\Rightarrow_d$*
(?ts-A[j:= (last-prog (hd-prog p$_j$ (Ghost$_{sb}$ A' L' R' W'# zs)) ys,
is$_j$',
j$_{sbj}$ |' (dom j$_{sbj}$ − read-tmps (Read$_{sb}$ False a' t v # zs)),(),
$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-Write$_{sb}$ ys ≠ {}, acquired True ys (acquired
True ?take-sb$_j$ $\mathcal{O}_j$),$\mathcal{R}_j$')],
flush ys ?m-A,
share ys ?share-A)
(**is** (-,-,-) $\Rightarrow_d$* (?ts-ys,?m-ys,?shared-ys))
by (auto)

note conflict-computation = rtranclp-trans [OF rtranclp-r-rtranclp [OF steps-flush-sb,
OF store-step] steps-ys]

from cph
have causal-program-history is$_{sbj}$ ((ys @ [Read$_{sb}$ False a' t v]) @ zs)
by simp
from causal-program-history-suffix [OF this]
have cph': causal-program-history is$_{sbj}$ zs.
interpret causal$_j$: causal-program-history is$_{sbj}$ zs **by** (rule cph')

from causal$_j$.causal-program-history [of [], simplified, OF refl] is$_j$'
obtain is$_j$''
where is$_j$': is$_j$' = Read False a' t#is$_j$'' **and**
is$_j$'': instrs zs @ is$_{sbj}$ = is$_j$'' @ prog-instrs zs
by clarsimp

from j-bound'''
have j-bound-ys: j < length ?ts-ys
by auto

from j-bound-ys neq-i-j
have ts-ys-j: ?ts-ys!j=(last-prog (hd-prog p$_j$ (Read$_{sb}$ False a' t v# zs)) ys, is$_j$',

j_{sbj} |‘ (dom j_{sbj} − read-tmps (Read$_{sb}$ False a′ t v# zs)),(),
$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-Write$_{sb}$ ys ≠ {},
acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$),$\mathcal{R}_j$′)
by auto

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe-delayed (?ts-ys,?m-ys,?shared-ys)**.**

from safe-delayedE [OF this j-bound-ys ts-ys-j, simplified is$_j$′]
have a′ ∈ acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ∨
a′ ∈ read-only (share ys (share ?drop-sb $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L))
apply cases
apply (auto simp add: Let-def is$_{sb}$)
done
with a′-unacq
have a′-ro: a′ ∈ read-only (share ys (share ?drop-sb $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L))
by auto
from a′-in
have a′-not-ro: a′ ∉ read-only (share ?drop-sb $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)
by (auto simp add: in-read-only-convs)

have a′ ∈ $\mathcal{O}_j$ ∪ all-acquired sb$_j$
proof −
{
assume a-notin: a′ ∉ $\mathcal{O}_j$ ∪ all-acquired sb$_j$
from weak-sharing-consis [OF j-bound″ ts$_{sb}$-j]
have weak-sharing-consistent $\mathcal{O}_j$ sb$_j$**.**
with weak-sharing-consistent-append [of $\mathcal{O}_j$?take-sb$_j$?drop-sb$_j$]
have weak-sharing-consistent (acquired True ?take-sb$_j$ $\mathcal{O}_j$) suspends$_j$
by (auto simp add: suspends$_j$)
with split-suspends$_j$
have weak-consis: weak-sharing-consistent (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ys
by (simp add: weak-sharing-consistent-append)
from all-acquired-append [of ?take-sb$_j$?drop-sb$_j$]
have all-acquired ys ⊆ all-acquired sb$_j$
apply (clarsimp)
apply (clarsimp simp add: suspends$_j$ [symmetric] split-suspends$_j$ all-acquired-append)
done
with a-notin acquired-takeWhile-non-volatile-Write$_{sb}$ [of sb$_j$ $\mathcal{O}_j$]
all-acquired-append [of ?take-sb$_j$?drop-sb$_j$]
have a′ ∉ acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$) $\mathcal{O}_j$ ∪ all-acquired ys
by auto

from read-only-share-unowned [OF weak-consis this a′-ro]
have a′ ∈ read-only (share ?drop-sb $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L) **.**

with a′-not-ro **have** False
by auto
}
thus ?thesis **by** blast

qed

moreover
from A-unaquired-by-others [rule-format, OF - neq-i-j] ts_{sb}-j j-bound
have A ∩ all-acquired sb_j = {}
by (auto simp add: Let-def)
moreover
from A-unowned-by-others [rule-format, OF - neq-i-j] ts_{sb}-j j-bound
have A ∩ $\mathcal{O}_j$ = {}
by (auto simp add: Let-def dest: all-shared-acquired-in)
moreover note a′-in
ultimately
show False
by auto
qed
}
thus ?thesis
by (auto simp add: Let-def)
qed

have valid-own′: valid-ownership $\mathcal{S}_{sb}'$ ts_{sb}'
proof (intro-locales)
show outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}_{sb}'$ ts_{sb}'
proof −
from outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ts_{sb}-i]
have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ (sb @ [$Ghost_{sb}$ A L R W])
by (auto simp add: non-volatile-owned-or-read-only-append)
from outstanding-non-volatile-refs-owned-or-read-only-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' sb′ $\mathcal{O}_{sb}'$ $\mathcal{S}_{sb}'$)
qed
next
show outstanding-volatile-writes-unowned-by-others ts_{sb}'
proof (unfold-locales)
fix i_1 j p_1 is_1 $\mathcal{O}_1$ $\mathcal{R}_1$ $\mathcal{D}_1$ xs_1 sb_1 p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs_j sb_j
assume i_1-bound: i_1 < length ts_{sb}'
assume j-bound: j < length ts_{sb}'
assume i_1-j: $i_1 \neq$ j
assume ts-i_1: ts_{sb}'!i_1 = (p_1,is_1,xs_1,sb_1,$\mathcal{D}_1$,$\mathcal{O}_1$,$\mathcal{R}_1$)
assume ts-j: ts_{sb}'!j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show ($\mathcal{O}_j$ ∪ all-acquired sb_j) ∩ outstanding-refs is-volatile-$Write_{sb}$ sb_1 = {}
proof (cases i_1=i)
case True
with i_1-j **have** i-j: i≠j
by simp

from j-bound **have** j-bound′: j < length ts_{sb}
by (simp add: ts_{sb}')
hence j-bound″: j < length (map owned ts_{sb})
by simp
from ts-j i-j **have** ts-j′: ts_{sb}!j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)

by (simp add: ts_{sb}')

from outstanding-volatile-writes-unowned-by-others
[OF i-bound j-bound' i-j ts_{sb}-i ts-j']
have ($\mathcal{O}_j$ ∪ all-acquired sb_j) ∩ outstanding-refs is-volatile-$Write_{sb}$ sb = {}.
with ts-i_1 True i-bound **show** ?thesis
by (clarsimp simp add: ts_{sb}' sb' outstanding-refs-append
acquired-takeWhile-non-volatile-$Write_{sb}$)
next
case False
note i_1-i = this
from i_1-bound **have** i_1-bound': i_1 < length ts_{sb}
by (simp add: ts_{sb}')
from ts-i_1 False **have** ts-i_1': ts_{sb}!i_1 = (p_1,is_1,xs_1,sb_1,$\mathcal{D}_1$,$\mathcal{O}_1$,$\mathcal{R}_1$)
by (simp add: ts_{sb}')
show ?thesis
proof (cases j=i)
case True

from i_1-bound'
have i_1-bound'': i_1 < length (map owned ts_{sb})
by simp

from outstanding-volatile-writes-unowned-by-others
[OF i_1-bound' i-bound i_1-i ts-i_1' ts_{sb}-i]
have ($\mathcal{O}_{sb}$ ∪ all-acquired sb) ∩ outstanding-refs is-volatile-$Write_{sb}$ sb_1 = {}.
moreover
from A-unused-by-others [rule-format, OF - False [symmetric]] False ts-i_1 i_1-bound
have A ∩ outstanding-refs is-volatile-$Write_{sb}$ sb_1 = {}
by (auto simp add: Let-def ts_{sb}')

ultimately
show ?thesis
using ts-j True ts_{sb}'
by (auto simp add: i-bound ts_{sb}' $\mathcal{O}_{sb}'$ sb' all-acquired-append)
next
case False
from j-bound **have** j-bound': j < length ts_{sb}
by (simp add: ts_{sb}')
from ts-j False **have** ts-j': ts_{sb}!j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (simp add: ts_{sb}')
from outstanding-volatile-writes-unowned-by-others
[OF i_1-bound' j-bound' i_1-j ts-i_1' ts-j']
show ($\mathcal{O}_j$ ∪ all-acquired sb_j) ∩ outstanding-refs is-volatile-$Write_{sb}$ sb_1 = {} .
qed
qed
qed
next
show read-only-reads-unowned ts_{sb}'
proof

fix n m
fix p_n is_n $\mathcal{O}_n$ $\mathcal{R}_n$ $\mathcal{D}_n$ j_n sb_n p_m is_m $\mathcal{O}_m$ $\mathcal{R}_m$ $\mathcal{D}_m$ j_m sb_m
assume n-bound: n < length ts_{sb}'
  and m-bound: m < length ts_{sb}'
  and neq-n-m: n≠m
  and nth: ts_{sb}'!n = (p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
  and mth: ts_{sb}'!m =(p_m, is_m, j_m, sb_m, $\mathcal{D}_m$, $\mathcal{O}_m$,$\mathcal{R}_m$)
from n-bound **have** n-bound′: n < length ts_{sb} **by** (simp add: ts_{sb}')
from m-bound **have** m-bound′: m < length ts_{sb} **by** (simp add: ts_{sb}')
show ($\mathcal{O}_m$ ∪ all-acquired sb_m) ∩
    read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_n) $\mathcal{O}_n$)
    (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb_n) =
    {}
proof (cases m=i)
  case True
  with neq-n-m **have** neq-n-i: n≠i
    by auto
  with n-bound nth i-bound **have** nth′: ts_{sb}!n =(p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
    by (auto simp add: ts_{sb}')
  note read-only-reads-unowned [OF n-bound′ i-bound neq-n-i nth′ ts_{sb}-i]
  moreover
  from A-no-read-only-reads-by-others [rule-format, OF - neq-n-i [symmetric]] n-bound′ nth′
  have A ∩ read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_n) $\mathcal{O}_n$)
      (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb_n) =
      {}
    by auto
  ultimately
  show ?thesis
    using True ts_{sb}-i nth′ mth n-bound′ m-bound′
    by (auto simp add: ts_{sb}' $\mathcal{O}_{sb}'$ sb′ all-acquired-append)
next
  case False
  note neq-m-i = this
  with m-bound mth i-bound **have** mth′: ts_{sb}!m = (p_m, is_m, j_m, sb_m, $\mathcal{D}_m$, $\mathcal{O}_m$,$\mathcal{R}_m$)
    by (auto simp add: ts_{sb}')
  show ?thesis
  proof (cases n=i)
    case True
      note read-only-reads-unowned [OF i-bound m-bound′ neq-m-i [symmetric] ts_{sb}-i mth′]
    then show ?thesis
using True neq-m-i ts_{sb}-i nth mth n-bound′ m-bound′
apply (case-tac outstanding-refs (is-volatile-Write$_{sb}$) sb = {})
apply (clarsimp simp add: outstanding-vol-write-take-drop-appends
  acquired-append read-only-reads-append ts_{sb}' sb′ $\mathcal{O}_{sb}'$)+
done
  next
    case False

with n-bound nth i-bound **have** nth′: ts_{sb}!n =(p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
by (auto simp add: ts_{sb}′)
from read-only-reads-unowned [OF n-bound′ m-bound′ neq-n-m nth′ mth′] False neq-m-i
show ?thesis
by (clarsimp)
qed
qed
qed
next
show ownership-distinct ts_{sb}′
proof −
have ∀j<length ts_{sb}. i ≠ j ⟶
(let (p_j, is_j,j_j, sb_j, $\mathcal{D}_j$, $\mathcal{O}_j$,$\mathcal{R}_j$) = ts_{sb} ! j
in ($\mathcal{O}_{sb}$ ∪ all-acquired sb′) ∩ ($\mathcal{O}_j$ ∪ all-acquired sb_j) = {})
proof −
{
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j
assume neq-i-j: i ≠ j
assume j-bound: j < length ts_{sb}
assume ts_{sb}-j: ts_{sb} ! j = (p_j, is_j, j_j, sb_j, $\mathcal{D}_j$, $\mathcal{O}_j$,$\mathcal{R}_j$)
have ($\mathcal{O}_{sb}$ ∪ all-acquired sb′) ∩ ($\mathcal{O}_j$ ∪ all-acquired sb_j) = {}
proof −
{
fix a′
assume a′-in-i: a′ ∈ ($\mathcal{O}_{sb}$ ∪ all-acquired sb′)
assume a′-in-j: a′ ∈ ($\mathcal{O}_j$ ∪ all-acquired sb_j)
have False
proof −
from a′-in-i **have** a′ ∈ ($\mathcal{O}_{sb}$ ∪ all-acquired sb) ∨ a′ ∈ A
by (simp add: sb′ all-acquired-append)
then show False
proof
assume a′ ∈ ($\mathcal{O}_{sb}$ ∪ all-acquired sb)
with ownership-distinct [OF i-bound j-bound neq-i-j ts_{sb}-i ts_{sb}-j] a′-in-j
show ?thesis
by auto
next
assume a′ ∈ A
moreover
have j-bound′: j < length (map owned ts_{sb})
using j-bound **by** auto
from A-unowned-by-others [rule-format, OF - neq-i-j] ts_{sb}-j j-bound
obtain A ∩ acquired True (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j) $\mathcal{O}_j$ = {} **and**
A ∩ all-shared (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j) = {}
by (auto simp add: Let-def)
moreover
from A-unaquired-by-others [rule-format, OF - neq-i-j] ts_{sb}-j j-bound
have A ∩ all-acquired sb_j = {}
by auto

ultimately
show ?thesis
using a′-in-j
by (auto dest: all-shared-acquired-in)
qed
qed
}
then show ?thesis **by** auto
qed
}
then show ?thesis **by** (fastforce simp add: Let-def)
qed

from ownership-distinct-nth-update [OF i-bound ts_{sb}-i this]
show ?thesis **by** (simp add: ts_{sb}' $\mathcal{O}_{sb}'$ sb′)
qed
qed

have valid-hist′: valid-history program-step ts_{sb}'
proof −
from valid-history [OF i-bound ts_{sb}-i]
have history-consistent j_{sb} (hd-prog p_{sb} sb) sb**.**
with valid-write-sops [OF i-bound ts_{sb}-i]
valid-implies-valid-prog-hd [OF i-bound ts_{sb}-i valid]
have history-consistent j_{sb} (hd-prog p_{sb} (sb@[$Ghost_{sb}$ A L R W]))
(sb@ [$Ghost_{sb}$ A L R W])
apply −
apply (rule history-consistent-appendI)
apply (auto simp add: hd-prog-append-$Ghost_{sb}$)
done
from valid-history-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' sb′ j_{sb}')
qed

have valid-reads′: valid-reads m_{sb} ts_{sb}'
proof −
from valid-reads [OF i-bound ts_{sb}-i]
have reads-consistent False $\mathcal{O}_{sb}$ m_{sb} sb **.**
from reads-consistent-snoc-$Ghost_{sb}$ [OF this]
have reads-consistent False $\mathcal{O}_{sb}$ m_{sb} (sb @ [$Ghost_{sb}$ A L R W])**.**
from valid-reads-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' sb′ $\mathcal{O}_{sb}'$)
qed

have valid-sharing′: valid-sharing $\mathcal{S}_{sb}'$ ts_{sb}'
proof (intro-locales)
from outstanding-non-volatile-writes-unshared [OF i-bound ts_{sb}-i]
have non-volatile-writes-unshared $\mathcal{S}_{sb}$ (sb @ [$Ghost_{sb}$ A L R W])
by (auto simp add: non-volatile-writes-unshared-append)
from outstanding-non-volatile-writes-unshared-nth-update [OF i-bound this]

show outstanding-non-volatile-writes-unshared $\mathcal{S}_{\mathsf{sb}}{}'$ $\mathrm{ts}_{\mathsf{sb}}{}'$
  by (simp add: $\mathrm{ts}_{\mathsf{sb}}{}'$ sb$'$ $\mathcal{S}_{\mathsf{sb}}{}'$)
    next
from sharing-consis [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i]
have consis$'$: sharing-consistent $\mathcal{S}_{\mathsf{sb}}$ $\mathcal{O}_{\mathsf{sb}}$ sb**.**
from A-shared-owned
have A $\subseteq$ dom (share ?drop-sb $\mathcal{S}$) $\cup$ acquired True sb $\mathcal{O}_{\mathsf{sb}}$
  by (simp add: sharing-consistent-append acquired-takeWhile-non-volatile-Write$_{\mathsf{sb}}$)
moreover have dom (share ?drop-sb $\mathcal{S}$) $\subseteq$ dom $\mathcal{S}$ $\cup$ dom (share sb $\mathcal{S}_{\mathsf{sb}}$)
proof
  fix a$'$
  assume a$'$-in: a$'$ $\in$ dom (share ?drop-sb $\mathcal{S}$)
  from share-unshared-in [OF a$'$-in]
  show a$'$ $\in$ dom $\mathcal{S}$ $\cup$ dom (share sb $\mathcal{S}_{\mathsf{sb}}$)
  proof
    assume a$'$ $\in$ dom (share ?drop-sb Map.empty)
    from share-mono-in [OF this] share-append [of ?take-sb ?drop-sb]
    have a$'$ $\in$ dom (share sb $\mathcal{S}_{\mathsf{sb}}$)
      by auto
    thus ?thesis
      by simp
  next
    assume a$'$ $\in$ dom $\mathcal{S}$ $\wedge$ a$'$ $\notin$ all-unshared ?drop-sb
    thus ?thesis **by** auto
  qed
qed
ultimately
have A-subset: A $\subseteq$ dom $\mathcal{S}$ $\cup$ dom (share sb $\mathcal{S}_{\mathsf{sb}}$) $\cup$ acquired True sb $\mathcal{O}_{\mathsf{sb}}$
  by auto
      have A $\subseteq$ dom (share sb $\mathcal{S}_{\mathsf{sb}}$) $\cup$ acquired True sb $\mathcal{O}_{\mathsf{sb}}$
      proof $-$
        {
          fix x
          assume x-A: x $\in$ A
          have x $\in$ dom (share sb $\mathcal{S}_{\mathsf{sb}}$) $\cup$ acquired True sb $\mathcal{O}_{\mathsf{sb}}$
          proof $-$
            {
              assume x $\in$ dom $\mathcal{S}$

            from share-all-until-volatile-write-share-acquired [OF ‹sharing-consis $\mathcal{S}_{\mathsf{sb}}$ $\mathrm{ts}_{\mathsf{sb}}$›

                i-bound $\mathrm{ts}_{\mathsf{sb}}$-i this [simplified $\mathcal{S}$]]
                A-unowned-by-others x-A
              have ?thesis
              by (fastforce simp add: Let-def)
            }
            with A-subset **show** ?thesis **using** x-A **by** auto
          qed
        }
        thus ?thesis **by** blast

qed
with consis′ L-subset A-R R-acq
have sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ (sb @ [Ghost$_{sb}$ A L R W])
by (simp add: sharing-consistent-append acquired-takeWhile-non-volatile-Write$_{sb}$)
from sharing-consis-nth-update [OF i-bound this]
show sharing-consis $\mathcal{S}_{sb}$′ ts$_{sb}$′
by (simp add: ts$_{sb}$′ $\mathcal{O}_{sb}$′ sb′ $\mathcal{S}_{sb}$′)

next
from read-only-unowned-nth-update [OF i-bound read-only-unowned [OF i-bound ts$_{sb}$-i]
]
show read-only-unowned $\mathcal{S}_{sb}$′ ts$_{sb}$′
by (simp add: $\mathcal{S}_{sb}$′ ts$_{sb}$′ $\mathcal{O}_{sb}$′)
next
from unowned-shared-nth-update [OF i-bound ts$_{sb}$-i subset-refl]
show unowned-shared $\mathcal{S}_{sb}$′ ts$_{sb}$′
by (simp add: ts$_{sb}$′ sb′ $\mathcal{O}_{sb}$′ $\mathcal{S}_{sb}$′)
next
from no-outstanding-write-to-read-only-memory [OF i-bound ts$_{sb}$-i]
have no-write-to-read-only-memory $\mathcal{S}_{sb}$ (sb @ [Ghost$_{sb}$ A L R W])
by (simp add: no-write-to-read-only-memory-append)

from no-outstanding-write-to-read-only-memory-nth-update [OF i-bound this]
show no-outstanding-write-to-read-only-memory $\mathcal{S}_{sb}$′ ts$_{sb}$′
by (simp add: $\mathcal{S}_{sb}$′ ts$_{sb}$′ sb′)
qed

have tmps-distinct′: tmps-distinct ts$_{sb}$′
proof (intro-locales)
from load-tmps-distinct [OF i-bound ts$_{sb}$-i]
have distinct-load-tmps is$_{sb}$′ **by** (simp add: is$_{sb}$)
from load-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-distinct ts$_{sb}$′ **by** (simp add: ts$_{sb}$′)
next
from read-tmps-distinct [OF i-bound ts$_{sb}$-i]
have distinct-read-tmps (sb @ [Ghost$_{sb}$ A L R W])
by (auto simp add: distinct-read-tmps-append)
from read-tmps-distinct-nth-update [OF i-bound this]
show read-tmps-distinct ts$_{sb}$′ **by** (simp add: ts$_{sb}$′ sb′)
next
from load-tmps-read-tmps-distinct [OF i-bound ts$_{sb}$-i]
have load-tmps is$_{sb}$′ ∩ read-tmps (sb @ [Ghost$_{sb}$ A L R W]) ={}
by (auto simp add: read-tmps-append is$_{sb}$)
from load-tmps-read-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-read-tmps-distinct ts$_{sb}$′ **by** (simp add: ts$_{sb}$′ sb′)
qed

have valid-sops′: valid-sops ts$_{sb}$′
proof −
from valid-store-sops [OF i-bound ts$_{sb}$-i]

obtain
  valid-store-sops′: ∀ sop∈store-sops is_{sb}′. valid-sop sop
  by (auto simp add: is_{sb})
from valid-write-sops [OF i-bound ts_{sb}-i]
have valid-write-sops′: ∀ sop∈write-sops (sb@ [$Ghost_{sb}$ A L R W]).
  valid-sop sop
  by (auto simp add: write-sops-append)
from valid-sops-nth-update [OF i-bound valid-write-sops′ valid-store-sops′]
show ?thesis **by** (simp add: ts_{sb}′ sb′)
    qed

    have valid-dd′: valid-data-dependency ts_{sb}′
    proof −
from data-dependency-consistent-instrs [OF i-bound ts_{sb}-i]
obtain
  dd-is: data-dependency-consistent-instrs (dom j_{sb}′) is_{sb}′
  by (auto simp add: is_{sb} j_{sb}′)
from load-tmps-write-tmps-distinct [OF i-bound ts_{sb}-i]
have load-tmps is_{sb}′ ∩ ⋃(fst ‘ write-sops (sb@ [$Ghost_{sb}$ A L R W])) ={}
  by (auto simp add: write-sops-append is_{sb})
from valid-data-dependency-nth-update [OF i-bound dd-is this]
show ?thesis **by** (simp add: ts_{sb}′ sb′)
    qed

    have load-tmps-fresh′: load-tmps-fresh ts_{sb}′
    proof −
from load-tmps-fresh [OF i-bound ts_{sb}-i]
have load-tmps is_{sb}′ ∩ dom j_{sb} = {}
  by (auto simp add: is_{sb})
from load-tmps-fresh-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}′ j_{sb}′)
    qed

    have enough-flushs′: enough-flushs ts_{sb}′
    proof −
from clean-no-outstanding-volatile-$Write_{sb}$ [OF i-bound ts_{sb}-i]
have ¬ $\mathcal{D}_{sb}$ ⟶ outstanding-refs is-volatile-$Write_{sb}$ (sb@[$Ghost_{sb}$ A L R W])= {}
  by (auto simp add: outstanding-refs-append)
from enough-flushs-nth-update [OF i-bound this]
show ?thesis
  by (simp add: ts_{sb}′ sb′ $\mathcal{D}_{sb}$′)
    qed

    have valid-program-history′: valid-program-history ts_{sb}′
    proof −
from valid-program-history [OF i-bound ts_{sb}-i]
have causal-program-history is_{sb} sb **.**
then have causal′: causal-program-history is_{sb}′ (sb@[$Ghost_{sb}$ A L R W])
  by (auto simp: causal-program-history-Ghost is_{sb})

from valid-last-prog [OF i-bound ts_{sb}-i]
have last-prog p_{sb} sb = p_{sb}.
hence last-prog p_{sb} (sb @ [$Ghost_{sb}$ A L R W]) = p_{sb}
  by (simp add: last-prog-append-$Ghost_{sb}$)
from valid-program-history-nth-update [OF i-bound causal′ this]
show ?thesis
  by (simp add: ts_{sb}′ sb′)
    qed

    show ?thesis
    proof (cases outstanding-refs is-volatile-$Write_{sb}$ sb = {})
case True

from True **have** flush-all: takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb = sb
  by (auto simp add: outstanding-refs-conv)

from True **have** suspend-nothing: dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb = []
  by (auto simp add: outstanding-refs-conv)

hence suspends-empty: suspends = []
  by (simp add: suspends)

from suspends-empty is-sim **have** is: is =Ghost A L R W# is_{sb}′
  by (simp add: is_{sb})

with suspends-empty ts-i
have ts-i: ts!i = (p_{sb}, Ghost A L R W# is_{sb}′,
  j_{sb},(), $\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$,release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
  by simp

from direct-memop-step.Ghost
have (Ghost A L R W# is_{sb}′,
  j_{sb}, (),m, $\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$,
      release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$, $\mathcal{S}$) →
      (is_{sb}′,
  j_{sb}, (), m, $\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$ ∪ A − R,
    augment-rels (dom $\mathcal{S}$) R (release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$),
    $\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L).
from direct-computation.concurrent-step.Memop [OF i-bound′ ts-i this]
have (ts, m, $\mathcal{S}$) $\Rightarrow_d$
      (ts[i := (p_{sb}, is_{sb}′,
    j_{sb}, (),$\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$ ∪ A − R,
      augment-rels (dom $\mathcal{S}$) R (release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$))],
  m,$\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L).

moreover

from suspend-nothing
have suspend-nothing′: (dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb′) = []
  by (simp add: sb′)

have all-shared-A: $\forall$j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts_{sb} $\longrightarrow$ i $\neq$ j $\longrightarrow$
ts_{sb} ! j = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-shared (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb) $\cap$ A = {}
proof –
{
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j x
assume j-bound: j < length ts_{sb}
assume neq-i-j: i $\neq$ j
assume jth: ts_{sb}!j = (p_j,is_j, j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
assume x-shared: x $\in$ all-shared (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
assume x-A: x $\in$ A
have False
proof –
from all-shared-acquired-or-owned [OF sharing-consis [OF j-bound jth]]
have all-shared sb_j $\subseteq$ all-acquired sb_j $\cup$ $\mathcal{O}_j$.

moreover have all-shared (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j) $\subseteq$ all-shared sb_j
using all-shared-append [of (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
(dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)]
by auto
moreover

from A-unaquired-by-others [rule-format, OF - neq-i-j] jth j-bound
have A $\cap$ all-acquired sb_j = {} **by** auto
moreover

from A-unowned-by-others [rule-format, OF - neq-i-j] jth j-bound
have A $\cap$ $\mathcal{O}_j$ = {}
by (auto dest: all-shared-acquired-in)

ultimately
show False
using x-A x-shared
by blast
qed
}
thus ?thesis **by** blast
qed

hence all-shared-L: $\forall$j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts_{sb} $\longrightarrow$ i $\neq$ j $\longrightarrow$
ts_{sb} ! j = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-shared (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb) $\cap$ L = {}
using L-subset **by** blast

have all-shared-A: $\forall$j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts_{sb} $\longrightarrow$ i $\neq$ j $\longrightarrow$
ts_{sb} ! j = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$

all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ A = {}
proof –
{
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j x
assume j-bound: j < length ts_{sb}
assume jth: ts_{sb}!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
assume neq-i-j: i ≠ j
assume x-shared: x ∈ all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)
assume x-A: x ∈ A
have False
proof –
from all-shared-acquired-or-owned [OF sharing-consis [OF j-bound jth]]
have all-shared sb_j ⊆ all-acquired sb_j ∪ $\mathcal{O}_j$.

moreover have all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j) ⊆ all-shared
sb_j
using all-shared-append [of (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb_j)]
by auto
moreover
from A-unaquired-by-others [rule-format, OF - neq-i-j] jth j-bound
have A ∩ all-acquired sb_j = {} **by** auto
moreover

from A-unowned-by-others [rule-format, OF - neq-i-j] jth j-bound
have A ∩ $\mathcal{O}_j$ = {}
by (auto dest: all-shared-acquired-in)

ultimately
show False
using x-A x-shared
by blast
qed
}
thus ?thesis **by** blast
qed
hence all-shared-L: ∀j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts_{sb} ⟶ i ≠ j ⟶
ts_{sb} ! j = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) ⟶
all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ L = {}
using L-subset **by** blast

have all-unshared-R: ∀j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts_{sb} ⟶ i ≠ j ⟶
ts_{sb} ! j = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) ⟶
all-unshared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∩ R = {}
proof –
{
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j x
assume j-bound: j < length ts_{sb}
assume neq-i-j: i ≠ j

assume jth: $ts_{sb}!j = (p_j,is_j,j_j,sb_j,\mathcal{D}_j,\mathcal{O}_j,\mathcal{R}_j)$
assume x-unshared: $x \in$ all-unshared (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j)
assume x-R: $x \in R$
have False
proof –
from unshared-acquired-or-owned [OF sharing-consis [OF j-bound jth]]
have all-unshared $sb_j \subseteq$ all-acquired $sb_j \cup \mathcal{O}_j$.

moreover have all-unshared (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j) $\subseteq$ all-unshared sb_j
using all-unshared-append [of (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j)
(dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j)]
by auto
moreover

note ownership-distinct [OF i-bound j-bound neq-i-j ts_{sb}-i jth]

ultimately
show False
using R-acq x-R x-unshared acquired-all-acquired [of True sb $\mathcal{O}_{sb}$]
by blast
qed
}
thus ?thesis **by** blast
qed

have all-acquired-R: $\forall$j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts_{sb} $\longrightarrow$ i $\neq$ j $\longrightarrow$
ts_{sb} ! j = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\cap$ R = {}
proof –
{
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j x
assume j-bound: j < length ts_{sb}
assume jth: $ts_{sb}!j = (p_j,is_j,j_j,sb_j,\mathcal{D}_j,\mathcal{O}_j,\mathcal{R}_j)$
assume neq-i-j: i $\neq$ j
assume x-acq: $x \in$ all-acquired (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j)
assume x-R: $x \in R$
have False
proof –

from x-acq **have** $x \in$ all-acquired sb_j
using all-acquired-append [of takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j
dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j]
by auto
moreover
note ownership-distinct [OF i-bound j-bound neq-i-j ts_{sb}-i jth]
ultimately
show False
using R-acq x-R acquired-all-acquired [of True sb $\mathcal{O}_{sb}$]
by blast

qed
}
thus ?thesis **by** blast
qed

have all-shared-R: $\forall$ j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb. j < length ts_{sb} $\longrightarrow$ i $\neq$ j $\longrightarrow$
ts_{sb} ! j = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) $\longrightarrow$
all-shared (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb) $\cap$ R = {}
proof –
{
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j x
assume j-bound: j < length ts_{sb}
assume jth: ts_{sb}!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
assume neq-i-j: i $\neq$ j
assume x-shared: x $\in$ all-shared (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
assume x-R: x $\in$ R
have False
proof –
from all-shared-acquired-or-owned [OF sharing-consis [OF j-bound jth]]
have all-shared sb_j $\subseteq$ all-acquired sb_j $\cup$ $\mathcal{O}_j$.

moreover have all-shared (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j) $\subseteq$ all-shared sb_j
using all-shared-append [of (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
(dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)]
by auto
moreover
note ownership-distinct [OF i-bound j-bound neq-i-j ts_{sb}-i jth]
ultimately
show False
using R-acq x-R x-shared acquired-all-acquired [of True sb $\mathcal{O}_{sb}$]
by blast
qed
}
thus ?thesis **by** blast
qed

note share-commute =
share-all-until-volatile-write-append-$Ghost_{sb}$ [OF True ‹ownership-distinct ts_{sb}› ‹sharing-consis $\mathcal{S}_{sb}$ ts_{sb}›
i-bound ts_{sb}-i all-shared-L all-shared-A all-acquired-R all-unshared-R all-shared-R]

from $\mathcal{D}$
have $\mathcal{D}'$: $\mathcal{D}_{sb}$ = ($\mathcal{D}$ $\vee$ outstanding-refs is-volatile-$Write_{sb}$ (sb@[$Ghost_{sb}$ A L R W]) $\neq$ {})
by (auto simp: outstanding-refs-append)

have $\forall$ a $\in$ R. (a $\in$ (dom (share sb $\mathcal{S}_{sb}$))) = (a $\in$ dom $\mathcal{S}$)
proof –
{

fix a
assume a-R: a ∈ R
have (a ∈ (dom (share sb $\mathcal{S}_{\mathsf{sb}}$))) = (a ∈ dom $\mathcal{S}$)
proof −
from a-R R-acq acquired-all-acquired [of True sb $\mathcal{O}_{\mathsf{sb}}$]
have a ∈ $\mathcal{O}_{\mathsf{sb}}$ ∪ all-acquired sb
by auto

from share-all-until-volatile-write-thread-local′ [OF ownership-distinct-ts$_{\mathsf{sb}}$ sharing-consis-ts$_{\mathsf{sb}}$ i-bound ts$_{\mathsf{sb}}$-i this] suspend-nothing
show ?thesis **by** (auto simp add: domIff $\mathcal{S}$)
qed
}
then show ?thesis **by** auto
qed
from augment-rels-shared-exchange [OF this]
have rel-commute:
augment-rels (dom $\mathcal{S}$) R (release sb (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}_{\mathsf{sb}}$) =
release (sb @ [Ghost$_{\mathsf{sb}}$ A L R W]) (dom $\mathcal{S}_{\mathsf{sb}}$′) $\mathcal{R}_{\mathsf{sb}}$
by (clarsimp simp add: release-append $\mathcal{S}_{\mathsf{sb}}$′)

have (ts$_{\mathsf{sb}}$′,m$_{\mathsf{sb}}$,$\mathcal{S}_{\mathsf{sb}}$′) ∼
(ts[i := (p$_{\mathsf{sb}}$,is$_{\mathsf{sb}}$′,
j$_{\mathsf{sb}}$,(), $\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{\mathsf{sb}}$ ∪ A − R,
augment-rels (dom $\mathcal{S}$) R (release ?take-sb (dom $\mathcal{S}_{\mathsf{sb}}$) $\mathcal{R}_{\mathsf{sb}}$))],
m,$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
apply (rule sim-config.intros)
apply (simp add: m ts$_{\mathsf{sb}}$′ $\mathcal{O}_{\mathsf{sb}}$′ sb′ j$_{\mathsf{sb}}$′ flush-all-until-volatile-write-append-Ghost-commute [OF i-bound ts$_{\mathsf{sb}}$-i])
apply (clarsimp simp add: $\mathcal{S}$ $\mathcal{S}_{\mathsf{sb}}$′ ts$_{\mathsf{sb}}$′ sb′ $\mathcal{O}_{\mathsf{sb}}$′ j$_{\mathsf{sb}}$′ share-commute)
using leq
apply (simp add: ts$_{\mathsf{sb}}$′)
using i-bound i-bound′ ts-sim ts-i True $\mathcal{D}$′
apply (clarsimp simp add: Let-def nth-list-update
outstanding-refs-conv ts$_{\mathsf{sb}}$′ $\mathcal{O}_{\mathsf{sb}}$′ $\mathcal{R}_{\mathsf{sb}}$′ $\mathcal{S}_{\mathsf{sb}}$′ j$_{\mathsf{sb}}$′ sb′ $\mathcal{D}_{\mathsf{sb}}$′ suspend-nothing′ flush-all rel-commute
acquired-append split: if-split-asm)
done

ultimately show ?thesis
using valid-own′ valid-hist′ valid-reads′ valid-sharing′ tmps-distinct′
valid-sops′
valid-dd′ load-tmps-fresh′ enough-flushs′
valid-program-history′ valid′ m$_{\mathsf{sb}}$′ $\mathcal{S}_{\mathsf{sb}}$′ $\mathcal{R}_{\mathsf{sb}}$′
by auto
next
case False

then obtain r **where** r-in: r ∈ set sb **and** volatile-r: is-volatile-Write$_{\mathsf{sb}}$ r

by (auto simp add: outstanding-refs-conv)
from takeWhile-dropWhile-real-prefix
[OF r-in, of (Not ∘ is-volatile-Write$_{sb}$), simplified, OF volatile-r]
obtain a′ v′ sb″ A″ L″ R″ W″ sop′ **where**
sb-split: sb = takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb @ Write$_{sb}$ True a′ sop′ v′ A″ L″ R″ W″# sb″
and
drop: dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb = Write$_{sb}$ True a′ sop′ v′ A″ L″ R″ W″# sb″
apply (auto)
subgoal for y ys
apply (case-tac y)
apply auto
done
done
from drop suspends **have** suspends: suspends = Write$_{sb}$ True a′ sop′ v′ A″ L″ R″ W″# sb″
by simp

have (ts, m, $\mathcal{S}$) $\Rightarrow_{\mathsf{d}}^*$ (ts, m, $\mathcal{S}$) **by** auto
moreover

have Write$_{sb}$ True a′ sop′ v′ A″ L″ R″ W″∈ set sb
by (subst sb-split) auto
note drop-app = dropWhile-append1
[OF this, of (Not ∘ is-volatile-Write$_{sb}$), simplified]

from takeWhile-append1 [**where** P=Not ∘ is-volatile-Write$_{sb}$, OF r-in] volatile-r
have takeWhile-app:
(takeWhile (Not ∘ is-volatile-Write$_{sb}$) (sb @ [Ghost$_{sb}$ A L R W])) = (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
by simp

note share-commute = share-all-until-volatile-write-append-Ghost$_{sb}$′ [OF False i-bound ts$_{sb}$-i]

from $\mathcal{D}$
have $\mathcal{D}$′: $\mathcal{D}_{sb}$ = ($\mathcal{D}$ ∨ outstanding-refs is-volatile-Write$_{sb}$ (sb@[Ghost$_{sb}$ A L R W]) ≠ {})
by (auto simp: outstanding-refs-append)

have (ts$_{sb}$′,m$_{sb}$,$\mathcal{S}_{sb}$′) ∼ (ts,m,$\mathcal{S}$)
apply (rule sim-config.intros)
apply (simp add: m flush-all-until-volatile-write-append-Ghost-commute [OF i-bound ts$_{sb}$-i] ts$_{sb}$′ $\mathcal{O}_{sb}$′ j$_{sb}$′ sb′)
apply (clarsimp simp add: $\mathcal{S}$ $\mathcal{S}_{sb}$′ ts$_{sb}$′ sb′ $\mathcal{O}_{sb}$′ j$_{sb}$′ share-commute)
using leq
apply (simp add: ts$_{sb}$′)
using i-bound i-bound′ ts-sim ts-i is-sim $\mathcal{D}$′
apply (clarsimp simp add: Let-def nth-list-update is-sim drop-app

read-tmps-append suspends
prog-instrs-append-Ghost$_{sb}$ instrs-append-Ghost$_{sb}$ hd-prog-append-Ghost$_{sb}$
drop is$_{sb}$ ts$_{sb}$' sb' $\mathcal{O}_{sb}$' $\mathcal{R}_{sb}$' $\mathcal{S}_{sb}$' j$_{sb}$' $\mathcal{D}_{sb}$' takeWhile-app split: if-split-asm)
done
ultimately show ?thesis
using valid-own' valid-hist' valid-reads' valid-sharing' tmps-distinct' valid-dd'
valid-sops' load-tmps-fresh' enough-flushs'
valid-program-history' valid' m$_{sb}$' $\mathcal{S}_{sb}$'
by (auto simp del: fun-upd-apply)
qed
qed
next
case (StoreBuffer i p$_{sb}$ is$_{sb}$ j$_{sb}$ sb $\mathcal{D}_{sb}$ $\mathcal{O}_{sb}$ $\mathcal{R}_{sb}$ sb' $\mathcal{O}_{sb}$' $\mathcal{R}_{sb}$')
then obtain

ts$_{sb}$': ts$_{sb}$' = ts$_{sb}$[i := (p$_{sb}$, is$_{sb}$, j$_{sb}$, sb', $\mathcal{D}_{sb}$, $\mathcal{O}_{sb}$',$\mathcal{R}_{sb}$')] **and**
i-bound: i < length ts$_{sb}$ **and**
ts$_{sb}$-i: ts$_{sb}$! i = (p$_{sb}$, is$_{sb}$, j$_{sb}$,sb, $\mathcal{D}_{sb}$, $\mathcal{O}_{sb}$,$\mathcal{R}_{sb}$) **and**
flush: (m$_{sb}$,sb,$\mathcal{O}_{sb}$,$\mathcal{R}_{sb}$,$\mathcal{S}_{sb}$) $\rightarrow_f$
(m$_{sb}$',sb',$\mathcal{O}_{sb}$',$\mathcal{R}_{sb}$',$\mathcal{S}_{sb}$')
by auto

from sim **obtain**
m: m = flush-all-until-volatile-write ts$_{sb}$ m$_{sb}$ **and**
$\mathcal{S}$: $\mathcal{S}$ = share-all-until-volatile-write ts$_{sb}$ $\mathcal{S}_{sb}$ **and**
leq: length ts$_{sb}$ = length ts **and**
ts-sim: $\forall$i<length ts$_{sb}$.
let (p, is$_{sb}$, j, sb,$\mathcal{D}_{sb}$, $\mathcal{O}_{sb}$,$\mathcal{R}$) = ts$_{sb}$! i;
suspends = dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb
in $\exists$is $\mathcal{D}$. instrs suspends @ is$_{sb}$ = is @ prog-instrs suspends $\wedge$
$\mathcal{D}_{sb}$ = ($\mathcal{D}$ $\vee$ outstanding-refs is-volatile-Write$_{sb}$ sb $\neq$ {}) $\wedge$
ts ! i =
(hd-prog p suspends,
is,
j |' (dom j − read-tmps suspends), (),
$\mathcal{D}$,
acquired True (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) $\mathcal{O}_{sb}$,
release (takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb) (dom $\mathcal{S}_{sb}$) $\mathcal{R}$)
by cases blast

from i-bound leq **have** i-bound': i < length ts
by auto

have split-sb: sb = takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb @ dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb
(**is** sb = ?take-sb@?drop-sb)
by simp

from ts-sim [rule-format, OF i-bound] ts$_{sb}$-i **obtain** suspends is $\mathcal{D}$ **where**

suspends: suspends = dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb **and**
is-sim: instrs suspends @ is$_{sb}$ = is @ prog-instrs suspends **and**
$\mathcal{D}$: $\mathcal{D}_{sb}$ = ($\mathcal{D}$ ∨ outstanding-refs is-volatile-Write$_{sb}$ sb ≠ {}) **and**
ts-i: ts ! i =
(hd-prog p$_{sb}$ suspends, is,
j$_{sb}$ |' (dom j$_{sb}$ − read-tmps suspends), (),$\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$,
release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
by (auto simp add: Let-def)

from flush-step-preserves-valid [OF i-bound ts$_{sb}$-i flush valid]
have valid′: valid ts$_{sb}$′
by (simp add: ts$_{sb}$′)

from flush **obtain** r **where** sb: sb=r#sb′
by (cases) auto

from valid-history [OF i-bound ts$_{sb}$-i]
have history-consistent j$_{sb}$ (hd-prog p$_{sb}$ sb) sb**.**
then
have hist-consis′: history-consistent j$_{sb}$ (hd-prog p$_{sb}$ sb′) sb′
by (auto simp add: sb intro: history-consistent-hd-prog
split: memref.splits option.splits)
from valid-history-nth-update [OF i-bound this]
have valid-hist′: valid-history program-step ts$_{sb}$′ **by** (simp add: ts$_{sb}$′)

from read-tmps-distinct [OF i-bound ts$_{sb}$-i]
have dist-sb′: distinct-read-tmps sb′
by (simp add: sb split: memref.splits)

have tmps-distinct′: tmps-distinct ts$_{sb}$′
proof (intro-locales)
from load-tmps-distinct [OF i-bound ts$_{sb}$-i]
have distinct-load-tmps is$_{sb}$**.**

from load-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-distinct ts$_{sb}$′
by (simp add: ts$_{sb}$′)
next
from read-tmps-distinct-nth-update [OF i-bound dist-sb′]
show read-tmps-distinct ts$_{sb}$′
by (simp add: ts$_{sb}$′)
next
from load-tmps-read-tmps-distinct [OF i-bound ts$_{sb}$-i]
have load-tmps is$_{sb}$ ∩ read-tmps sb′ = {}
by (auto simp add: sb split: memref.splits)
from load-tmps-read-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-read-tmps-distinct ts$_{sb}$′ **by** (simp add: ts$_{sb}$′)
qed

from load-tmps-write-tmps-distinct [OF i-bound ts$_{sb}$-i]

have load-tmps $is_{sb} \cap \bigcup$(fst ‘ write-sops sb′) = {}
by (auto simp add: sb split: memref.splits)
from valid-data-dependency-nth-update
[OF i-bound data-dependency-consistent-instrs [OF i-bound ts_{sb}-i] this]
have valid-dd′: valid-data-dependency ts_{sb}'
by (simp add: ts_{sb}')

from valid-store-sops [OF i-bound ts_{sb}-i] valid-write-sops [OF i-bound ts_{sb}-i]
valid-sops-nth-update [OF i-bound]
have valid-sops′: valid-sops ts_{sb}'
by (cases r) (auto simp add: sb ts_{sb}')

have load-tmps-fresh′: load-tmps-fresh ts_{sb}'
proof –
from load-tmps-fresh [OF i-bound ts_{sb}-i]
have load-tmps $is_{sb} \cap$ dom j_{sb} = {}.
from load-tmps-fresh-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}')
qed

have enough-flushs′: enough-flushs ts_{sb}'
proof –
from clean-no-outstanding-volatile-$Write_{sb}$ [OF i-bound ts_{sb}-i]
have $\neg \mathcal{D}_{sb} \longrightarrow$ outstanding-refs is-volatile-$Write_{sb}$ sb′ = {}
by (auto simp add: sb split: if-split-asm)
from enough-flushs-nth-update [OF i-bound this]
show ?thesis
by (simp add: ts_{sb}' sb)
qed

show ?thesis
proof (cases r)
case ($Write_{sb}$ volatile a sop v A L R W)
from flush this
have m_{sb}': $m_{sb}' = (m_{sb}(a := v))$
by cases (auto simp add: sb)

have non-volatile-owned: $\neg$ volatile $\longrightarrow a \in \mathcal{O}_{sb}$
proof (cases volatile)
case True **thus** ?thesis **by** simp
next
case False
with outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ts_{sb}-i]
have $a \in \mathcal{O}_{sb}$
by (simp add: sb $Write_{sb}$)
thus ?thesis **by** simp
qed

have a-unowned-by-others:
$\forall j <$ length ts_{sb}. $i \neq j \longrightarrow$ (let (-,-,-,sb_j,-,$\mathcal{O}_j$,-) = ts_{sb} ! j in

$a \notin \mathcal{O}_j \cup$ all-acquired sb_j)
proof (unfold Let-def, clarify del: notI)
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j
assume j-bound: $j <$ length ts_{sb}
assume neq: $i \neq j$
assume ts-j: ts_{sb} ! $j = (p_j, is_j, j_j, sb_j, \mathcal{D}_j, \mathcal{O}_j, \mathcal{R}_j)$
show $a \notin \mathcal{O}_j \cup$ all-acquired sb_j
proof (cases volatile)
case True
from outstanding-volatile-writes-unowned-by-others [OF i-bound j-bound neq ts_{sb}-i ts-j]
show ?thesis
by (simp add: sb $Write_{sb}$ True)
next
case False
with non-volatile-owned
have $a \in \mathcal{O}_{sb}$
by simp
with ownership-distinct [OF i-bound j-bound neq ts_{sb}-i ts-j]
show ?thesis
by blast
qed
qed

from valid-reads [OF i-bound ts_{sb}-i]
have reads-consis: reads-consistent False $\mathcal{O}_{sb}$ m_{sb} sb **.**

{
fix j
fix p_j is_{sbj} $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_{sbj}$ j_j sb_j
assume j-bound: $j <$ length ts_{sb}
assume ts_{sb}-j: ts_{sb}!j=$(p_j, is_{sbj}, j_j, sb_j, \mathcal{D}_{sbj}, \mathcal{O}_j, \mathcal{R}_j)$
assume neq-i-j: $i \neq j$
have $a \notin$ outstanding-refs is-$Write_{sb}$ (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
proof
assume $a \in$ outstanding-refs is-$Write_{sb}$ (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
hence $a \in$ outstanding-refs is-non-volatile-$Write_{sb}$ (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)
by (simp add: outstanding-refs-is-non-volatile-$Write_{sb}$-takeWhile-conv)
hence $a \in$ outstanding-refs is-non-volatile-$Write_{sb}$ sb_j
using outstanding-refs-append [of - (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j) (dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j)]
by auto
with non-volatile-owned-or-read-only-outstanding-non-volatile-writes
[OF outstanding-non-volatile-refs-owned-or-read-only [OF j-bound ts_{sb}-j]]
have $a \in \mathcal{O}_j \cup$ all-acquired sb_j
by auto
with a-unowned-by-others [rule-format, OF j-bound neq-i-j] ts_{sb}-j
show False

by auto
qed
}
note a-notin-others = this

from a-notin-others
have a-notin-others′:
$\forall$ j < length ts_{sb}. i $\neq$ j $\longrightarrow$
(let (-,-,-,sb_j,-,-,-) = ts_{sb}!j in a $\notin$ outstanding-refs is-$Write_{sb}$ (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j))
by (fastforce simp add: Let-def)

obtain D f **where** sop: sop=(D,f) **by** (cases sop) auto
from valid-history [OF i-bound ts_{sb}-i] sop sb $Write_{sb}$
obtain D-tmps: D $\subseteq$ dom j_{sb} **and** f-v: f j_{sb} = v **and**
D-sb′: D $\cap$ read-tmps sb′ = {}
by auto
let ?j = (j_{sb} |‘ (dom j_{sb} − read-tmps sb′))
from D-tmps D-sb′
have D-tmps′: D $\subseteq$ dom ?j
by auto
from valid-write-sops [OF i-bound ts_{sb}-i, rule-format, of sop]
have valid-sop sop
by (auto simp add: sb $Write_{sb}$)
from this [simplified sop]
interpret valid-sop (D,f) **.**
from D-tmps D-sb′
have ((dom j_{sb} − read-tmps sb′) $\cap$ D) = D
by blast
with valid-sop [OF refl D-tmps] valid-sop [OF refl D-tmps′] f-v
have f-v′: f ?j = v
by auto

have valid-program-history′: valid-program-history ts_{sb}′
proof −
from valid-program-history [OF i-bound ts_{sb}-i]
have causal-program-history is_{sb} sb **.**
then have causal′: causal-program-history is_{sb} sb′
by (simp add: sb $Write_{sb}$ causal-program-history-def)

from valid-last-prog [OF i-bound ts_{sb}-i]
have last-prog p_{sb} sb = p_{sb}**.**
hence last-prog p_{sb} sb′ = p_{sb}
by (simp add: sb $Write_{sb}$)

from valid-program-history-nth-update [OF i-bound causal′ this]
show ?thesis

by (simp add: ts_{sb}')
qed

show ?thesis
proof (cases volatile)
case True
note volatile = this
from flush Write_{sb} volatile
obtain
$\mathcal{O}_{sb}'$: $\mathcal{O}_{sb}'=\mathcal{O}_{sb} \cup A - R$ **and**
$\mathcal{S}_{sb}'$: $\mathcal{S}_{sb}'= \mathcal{S}_{sb} \oplus_W R \ominus_A L$ **and**
$\mathcal{R}_{sb}'$: $\mathcal{R}_{sb}'$ = Map.empty
by cases (auto simp add: sb)

from sharing-consis [OF i-bound ts_{sb}-i]
obtain
A-shared-owned: $A \subseteq \mathrm{dom}\ \mathcal{S}_{sb} \cup \mathcal{O}_{sb}$ **and**
L-subset: $L \subseteq A$ **and**
A-R: $A \cap R = \{\}$ **and**
R-owned: $R \subseteq \mathcal{O}_{sb}$
by (clarsimp simp add: sb Write_{sb} volatile)

from sb Write_{sb} True **have** take-empty: takeWhile (Not $\circ$ is-volatile-Write_{sb}) sb = []
by (auto simp add: outstanding-refs-conv)

from sb Write_{sb} True **have** suspend-all: dropWhile (Not $\circ$ is-volatile-Write_{sb}) sb = sb
by (auto simp add: outstanding-refs-conv)

hence suspends-all: suspends = sb
by (simp add: suspends)

from is-sim
have is-sim: Write True a (D, f) A L R W# instrs sb' @ is_{sb} = is @ prog-instrs sb'
by (simp add: True Write_{sb} suspends-all sb sop)

from valid-program-history [OF i-bound ts_{sb}-i]
interpret causal-program-history is_{sb} sb **.**
from valid-last-prog [OF i-bound ts_{sb}-i]
have last-prog: last-prog p_{sb} sb = p_{sb}**.**

from causal-program-history [of [Write_{sb} True a (D, f) v A L R W] sb'] is-sim
obtain is' **where**
is: is = Write True a (D, f) A L R W# is' **and**
is'-sim: instrs sb'@is_{sb} = is' @ prog-instrs sb'
by (auto simp add: sb Write_{sb} volatile sop)

from causal-program-history **have**
causal-program-history-sb′: causal-program-history is_{sb} sb′
apply −
apply (rule causal-program-history.intro)
apply (auto simp add: sb $Write_{sb}$)
done

from ts-i **have** ts-i: ts ! i =
(hd-prog p_{sb} sb′, Write True a (D, f) A L R W# is′, ?j, (), $\mathcal{D}$,acquired True ?take-sb $\mathcal{O}_{sb}$,
release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
by (simp add: suspends-all sb $Write_{sb}$ is)

let ?ts′ = ts[i := (hd-prog p_{sb} sb′, is′, ?j, (), True, acquired True ?take-sb $\mathcal{O}_{sb}$ ∪ A − R,
Map.empty)]

from i-bound′ **have** ts′-i: ?ts′!i = (hd-prog p_{sb} sb′, is′, ?j, (),True, acquired True ?take-sb $\mathcal{O}_{sb}$ ∪ A − R,Map.empty)
by simp

from no-outstanding-write-to-read-only-memory [OF i-bound ts_{sb}-i]
have a-not-ro: a ∉ read-only $\mathcal{S}_{sb}$
by (clarsimp simp add: sb $Write_{sb}$ volatile)

{
fix j
fix p_j is_{sbj} $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_{sbj}$ j_j sb_j
assume j-bound: j < length ts_{sb}
assume ts_{sb}-j: ts_{sb}!j=(p_j,is_{sbj},j_j,sb_j,$\mathcal{D}_{sbj}$,$\mathcal{O}_j$,$\mathcal{R}_j$)
assume neq-i-j: i≠j
have a ∉ unforwarded-non-volatile-reads (dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j) {}
proof
let ?take-sb_j = takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j
let ?drop-sb_j = dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j
assume a-in: a ∈ unforwarded-non-volatile-reads ?drop-sb_j {}

from a-unowned-by-others [rule-format, OF j-bound neq-i-j] ts_{sb}-j
obtain a-unowned: a ∉ $\mathcal{O}_j$ **and** a-unacq: a ∉ all-acquired sb_j
by auto
with all-acquired-append [of ?take-sb_j ?drop-sb_j] acquired-takeWhile-non-volatile-$Write_{sb}$ [of sb_j $\mathcal{O}_j$]
have a-unacq-take: a ∉ acquired True ?take-sb_j $\mathcal{O}_j$
by (auto simp add:)

note nvo-j = outstanding-non-volatile-refs-owned-or-read-only [OF j-bound ts_{sb}-j]

from non-volatile-owned-or-read-only-drop [OF nvo-j]
have nvo-drop-j: non-volatile-owned-or-read-only True (share ?take-sb_j $\mathcal{S}_{sb}$)
(acquired True ?take-sb_j $\mathcal{O}_j$) ?drop-sb_j **.**

note consis-j = sharing-consis [OF j-bound ts_{sb}-j]
with sharing-consistent-append [of $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb_j ?drop-sb_j]
obtain consis-take-j: sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_j$?take-sb_j **and**
consis-drop-j: sharing-consistent (share ?take-sb_j $\mathcal{S}_{sb}$)
(acquired True ?take-sb_j $\mathcal{O}_j$) ?drop-sb_j
by auto

from in-unforwarded-non-volatile-reads-non-volatile-$Read_{sb}$ [OF a-in]
have a-in′: a ∈ outstanding-refs is-non-volatile-$Read_{sb}$?drop-sb_j.

note reads-consis-j = valid-reads [OF j-bound ts_{sb}-j]
from reads-consistent-drop [OF this]
have reads-consis-drop-j:
reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) (flush ?take-sb_j m_{sb}) ?drop-sb_j.

from read-only-share-all-shared [of a ?take-sb_j $\mathcal{S}_{sb}$] a-not-ro
all-shared-acquired-or-owned [OF consis-take-j]
all-acquired-append [of ?take-sb_j ?drop-sb_j] a-unowned a-unacq
have a-not-ro-j: a ∉ read-only (share ?take-sb_j $\mathcal{S}_{sb}$)
by auto

from ts-sim [rule-format, OF j-bound] ts_{sb}-j j-bound
obtain $suspends_j$ is_j $\mathcal{D}_j$ $\mathcal{R}_j$ **where**
$suspends_j$: $suspends_j$ = ?drop-sb_j **and**
is_j: instrs $suspends_j$ @ is_{sbj} = is_j @ prog-instrs $suspends_j$ **and**
$\mathcal{D}_j$: $\mathcal{D}_{sbj}$ = ($\mathcal{D}_j$ ∨ outstanding-refs is-volatile-$Write_{sb}$ sb_j ≠ {}) **and**
ts_j: ts!j = (hd-prog p_j $suspends_j$, is_j,
j_j |‘ (dom j_j − read-tmps $suspends_j$),(),
$\mathcal{D}_j$, acquired True ?take-sb_j $\mathcal{O}_j$,$\mathcal{R}_j$)
by (auto simp: Let-def)

from valid-last-prog [OF j-bound ts_{sb}-j] **have** last-prog: last-prog p_j sb_j = p_j.

from j-bound i-bound′ leq **have** j-bound-ts′: j < length ts
by simp
from read-only-read-acquired-unforwarded-acquire-witness [OF nvo-drop-j consis-drop-j
a-not-ro-j a-unacq-take a-in]

have False
proof
assume ∃sop a′ v ys zs A L R W.
?drop-sb_j = ys @ $Write_{sb}$ True a′ sop v A L R W # zs ∧ a ∈ A ∧
a ∉ outstanding-refs is-$Write_{sb}$ ys ∧ a′≠a
with $suspends_j$
obtain a′ sop′ v′ ys zs′ A′ L′ R′ W′ **where**

split-suspends$_j$: suspends$_j$ = ys @ Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′# zs′ (**is** suspends$_j$=?suspends) **and**
a-A′: a ∈ A′ **and**
no-write: a ∉ outstanding-refs is-Write$_{sb}$ (ys @ [Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′])
by (auto simp add: outstanding-refs-append)

from last-prog
have lp: last-prog p$_j$ suspends$_j$ = p$_j$
apply −
apply (rule last-prog-same-append [**where** sb=?take-sb$_j$])
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
apply simp
done

from sharing-consis [OF j-bound ts$_{sb}$-j]
have sharing-consis-j: sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb$_j$.
then have A′-R′: A′ ∩ R′ = {}
by (simp add: sharing-consistent-append [of - - ?take-sb$_j$?drop-sb$_j$, simplified]
suspends$_j$ [symmetric] split-suspends$_j$ sharing-consistent-append)

from valid-program-history [OF j-bound ts$_{sb}$-j]
have causal-program-history is$_{sbj}$ sb$_j$.
then have cph: causal-program-history is$_{sbj}$?suspends
apply −
apply (rule causal-program-history-suffix [**where** sb=?take-sb$_j$])
apply (simp only: split-suspends$_j$ [symmetric] suspends$_j$)
apply (simp add: split-suspends$_j$)
done

from valid-reads [OF j-bound ts$_{sb}$-j]
have reads-consis-j: reads-consistent False $\mathcal{O}_j$ m$_{sb}$ sb$_j$.

from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts$_{sb}$›
j-bound ts$_{sb}$-j this]
have reads-consis-m-j: reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$) m suspends$_j$
by (simp add: m suspends$_j$)

hence reads-consis-ys: reads-consistent True (acquired True ?take-sb$_j$ $\mathcal{O}_j$)
m (ys@[Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′])
by (simp add: split-suspends$_j$ reads-consistent-append)

from valid-write-sops [OF j-bound ts$_{sb}$-j]
have ∀ sop∈write-sops (?take-sb$_j$@?suspends). valid-sop sop
by (simp add: split-suspends$_j$ [symmetric] suspends$_j$)
then obtain valid-sops-take: ∀ sop∈write-sops ?take-sb$_j$. valid-sop sop **and**
valid-sops-drop: ∀ sop∈write-sops (ys@[Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]). valid-sop sop
apply (simp only: write-sops-append)
apply auto

done

from read-tmps-distinct [OF j-bound ts_{sb}-j]
have distinct-read-tmps (?take-sb_j@$suspends_j$)
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb_j $\cap$ read-tmps $suspends_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps $suspends_j$
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp only: distinct-read-tmps-append)
done

from valid-history [OF j-bound ts_{sb}-j]
have h-consis:
history-consistent j_j (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof −
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop
h-consis] last-prog-hd-prog
have hist-consis′: history-consistent j_j (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis-j]
have no-vol-read: outstanding-refs is-volatile-$Read_{sb}$
(ys@[$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]) = {}
by (auto simp add: outstanding-refs-append $suspends_j$ [symmetric]
split-$suspends_j$)

have acq-simp:
acquired True (ys @ [$Write_{sb}$ True a′ sop′ v′ A′ L′ R′ W′])
(acquired True ?take-sb_j $\mathcal{O}_j$) =
acquired True ys (acquired True ?take-sb_j $\mathcal{O}_j$) $\cup$ A′ − R′

by (simp add: acquired-append)

from flush-store-buffer-append [**where** sb=ys@[Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′] **and** sb′=zs′, simplified,
OF j-bound-ts′ is$_j$ [simplified split-suspends$_j$] cph [simplified suspends$_j$] ts$_j$ [simplified split-suspends$_j$]
refl lp [simplified split-suspends$_j$] reads-consis-ys
hist-consis′ [simplified split-suspends$_j$] valid-sops-drop
distinct-read-tmps-drop [simplified split-suspends$_j$]
no-volatile-Read$_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=$\mathcal{S}$]

obtain is$_j$′ $\mathcal{R}_j$′ **where**
is$_j$′: instrs zs′ @ is$_{sbj}$ = is$_j$′ @ prog-instrs zs′ **and**
steps-ys: (ts, m, $\mathcal{S}$) $\Rightarrow_d^*$
(ts[j:=(last-prog
(hd-prog p$_j$ (Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′# zs′)) (ys@[Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]),
is$_j$′,
j$_j$ |‘ (dom j$_j$ − read-tmps zs′),
(), True, acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ∪ A′ − R′,$\mathcal{R}_j$′)],
flush (ys@[Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]) m,
share (ys@[Write$_{sb}$ True a′ sop′ v′ A′ L′ R′ W′]) $\mathcal{S}$)
(**is** (-,-,-) $\Rightarrow_d^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto simp add: acquired-append outstanding-refs-append)

from i-bound′ **have** i-bound-ys: i < length ?ts-ys
by auto

from i-bound′ neq-i-j ts-i
have ts-ys-i: ?ts-ys!i = (hd-prog p$_{sb}$ sb′, Write True a (D, f) A L R W# is′, ?j, (), $\mathcal{D}$,
acquired True ?take-sb $\mathcal{O}_{sb}$,release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
by simp

note conflict-computation = steps-ys

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe: safe-delayed (?ts-ys,?m-ys,?shared-ys).

with safe-delayedE [OF safe i-bound-ys ts-ys-i]
have a-unowned:
∀j < length ?ts-ys. i≠j ⟶ (let ($\mathcal{O}_j$) = map owned ?ts-ys!j in a ∉ $\mathcal{O}_j$)
apply cases
apply (auto simp add: Let-def sb)
done
from a-A′ a-unowned [rule-format, of j] neq-i-j j-bound leq A′-R′
show False
by (auto simp add: Let-def)

next
assume $\exists$A L R W ys zs. ?drop-sb_j = ys @ $Ghost_{sb}$ A L R W# zs $\wedge$ a $\in$ A $\wedge$ a $\notin$ outstanding-refs is-$Write_{sb}$ ys
with $suspends_j$
obtain ys zs$'$ A$'$ L$'$ R$'$ W$'$ **where**
split-$suspends_j$: $suspends_j$ = ys @ $Ghost_{sb}$ A$'$ L$'$ R$'$ W$'$# zs$'$ (**is** $suspends_j$=?suspends) **and**
a-A$'$: a $\in$ A$'$ **and**
no-write: a $\notin$ outstanding-refs is-$Write_{sb}$ (ys @ [$Ghost_{sb}$ A$'$ L$'$ R$'$ W$'$])
by (auto simp add: outstanding-refs-append)

from last-prog
have lp: last-prog p_j $suspends_j$ = p_j
apply –
apply (rule last-prog-same-append [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

from valid-program-history [OF j-bound ts_{sb}-j]
have causal-program-history is_{sbj} sb_j.
then have cph: causal-program-history is_{sbj} ?suspends
apply –
apply (rule causal-program-history-suffix [**where** sb=?take-sb_j])
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp add: split-$suspends_j$)
done

from valid-reads [OF j-bound ts_{sb}-j]
have reads-consis-j: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j.

from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts_{sb}›
j-bound ts_{sb}-j this]
have reads-consis-m-j: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$) m $suspends_j$
by (simp add: m $suspends_j$)

hence reads-consis-ys: reads-consistent True (acquired True ?take-sb_j $\mathcal{O}_j$)
m (ys@[$Ghost_{sb}$ A$'$ L$'$ R$'$ W$'$])
by (simp add: split-$suspends_j$ reads-consistent-append)

from valid-write-sops [OF j-bound ts_{sb}-j]
have $\forall$sop$\in$write-sops (?take-sb_j@?suspends). valid-sop sop
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain valid-sops-take: $\forall$sop$\in$write-sops ?take-sb_j. valid-sop sop **and**
valid-sops-drop: $\forall$sop$\in$write-sops (ys@[$Ghost_{sb}$ A$'$ L$'$ R$'$ W$'$]). valid-sop sop
apply (simp only: write-sops-append)
apply auto

done

from read-tmps-distinct [OF j-bound ts_{sb}-j]
have distinct-read-tmps (?take-sb_j@$suspends_j$)
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
then obtain
read-tmps-take-drop: read-tmps ?take-sb_j $\cap$ read-tmps $suspends_j$ = {} **and**
distinct-read-tmps-drop: distinct-read-tmps $suspends_j$
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply (simp only: distinct-read-tmps-append)
done

from valid-history [OF j-bound ts_{sb}-j]
have h-consis:
history-consistent j_j (hd-prog p_j (?take-sb_j@$suspends_j$)) (?take-sb_j@$suspends_j$)
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
apply simp
done

from sharing-consis [OF j-bound ts_{sb}-j]
have sharing-consis-j: sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb_j.
then have A′-R′: A′ $\cap$ R′ = {}
by (simp add: sharing-consistent-append [of - - ?take-sb_j ?drop-sb_j, simplified]
$suspends_j$ [symmetric] split-$suspends_j$ sharing-consistent-append)

have last-prog-hd-prog: last-prog (hd-prog p_j sb_j) ?take-sb_j = (hd-prog p_j $suspends_j$)
proof −
from last-prog **have** last-prog p_j (?take-sb_j@?drop-sb_j) = p_j
by simp
from last-prog-hd-prog-append′ [OF h-consis] this
have last-prog (hd-prog p_j $suspends_j$) ?take-sb_j = hd-prog p_j $suspends_j$
by (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
moreover
have last-prog (hd-prog p_j (?take-sb_j @ $suspends_j$)) ?take-sb_j =
last-prog (hd-prog p_j $suspends_j$) ?take-sb_j
apply (simp only: split-$suspends_j$ [symmetric] $suspends_j$)
by (rule last-prog-hd-prog-append)
ultimately show ?thesis
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
qed

from history-consistent-appendD [OF valid-sops-take read-tmps-take-drop
h-consis] last-prog-hd-prog
have hist-consis′: history-consistent j_j (hd-prog p_j $suspends_j$) $suspends_j$
by (simp add: split-$suspends_j$ [symmetric] $suspends_j$)
from reads-consistent-drop-volatile-writes-no-volatile-reads
[OF reads-consis-j]
have no-vol-read: outstanding-refs is-volatile-$Read_{sb}$
(ys@[$Ghost_{sb}$ A′ L′ R′ W′]) = {}
by (auto simp add: outstanding-refs-append $suspends_j$ [symmetric]

split-suspends$_j$)

have acq-simp:
acquired True (ys @ [Ghost$_{sb}$ A′ L′ R′ W′])
(acquired True ?take-sb$_j$ $\mathcal{O}_j$) =
acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ∪ A′ − R′
by (simp add: acquired-append)

from flush-store-buffer-append [**where** sb=ys@[Ghost$_{sb}$ A′ L′ R′ W′] **and** sb′=zs′, simplified,
OF j-bound-ts′ is$_j$ [simplified split-suspends$_j$] cph [simplified suspends$_j$]
ts$_j$ [simplified split-suspends$_j$] refl lp [simplified split-suspends$_j$] reads-consis-ys
hist-consis′ [simplified split-suspends$_j$] valid-sops-drop
distinct-read-tmps-drop [simplified split-suspends$_j$]
no-volatile-Read$_{sb}$-volatile-reads-consistent [OF no-vol-read], **where**
$\mathcal{S}$=$\mathcal{S}$]

obtain is$_j$′ $\mathcal{R}_j$′ **where**
is$_j$′: instrs zs′ @ is$_{sbj}$ = is$_j$′ @ prog-instrs zs′ **and**
steps-ys: (ts, m,$\mathcal{S}$) $\Rightarrow_d^*$
(ts[j:=(last-prog
(hd-prog p$_j$ (Ghost$_{sb}$ A′ L′ R′ W′# zs′)) (ys@[Ghost$_{sb}$ A′ L′ R′ W′]),
is$_j$′,
j$_j$ |‘ (dom j$_j$ − read-tmps zs′),
(),
$\mathcal{D}_j$ ∨ outstanding-refs is-volatile-Write$_{sb}$ ys ≠ {}, acquired True ys (acquired True ?take-sb$_j$ $\mathcal{O}_j$) ∪ A′ − R′,$\mathcal{R}_j$′)],
flush (ys@[Ghost$_{sb}$ A′ L′ R′ W′]) m, share (ys@[Ghost$_{sb}$ A′ L′ R′ W′]) $\mathcal{S}$)
(**is** (-,-,-) $\Rightarrow_d^*$ (?ts-ys,?m-ys,?shared-ys))
by (auto simp add: acquired-append outstanding-refs-append)

from i-bound′ **have** i-bound-ys: i < length ?ts-ys
by auto

from i-bound′ neq-i-j ts-i
have ts-ys-i: ?ts-ys!i = (hd-prog p$_{sb}$ sb′, Write True a (D, f) A L R W# is′, ?j, (), $\mathcal{D}$,
acquired True ?take-sb $\mathcal{O}_{sb}$,release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
by simp

note conflict-computation = steps-ys

from safe-reach-safe-rtrancl [OF safe-reach conflict-computation]
have safe: safe-delayed (?ts-ys,?m-ys,?shared-ys)**.**

with safe-delayedE [OF safe i-bound-ys ts-ys-i]
have a-unowned:

∀ j < length ?ts-ys. i≠j ⟶ (let ($\mathcal{O}_j$) = map owned ?ts-ys!j in a ∉ $\mathcal{O}_j$)

apply cases
apply (auto simp add: Let-def sb)
done
from a-A′ a-unowned [rule-format, of j] neq-i-j j-bound leq A′-R′
show False
by (auto simp add: Let-def)
qed
then show False
by simp
qed
}
note a-notin-unforwarded-non-volatile-reads-drop = this

have valid-reads′: valid-reads m_{sb}′ ts_{sb}′
proof (unfold-locales)
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j
assume j-bound: j < length ts_{sb}′
assume ts-j: ts_{sb}′!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show reads-consistent False $\mathcal{O}_j$ m_{sb}′ sb_j
proof (cases i=j)
case True
from reads-consis ts-j j-bound sb **show** ?thesis
by (clarsimp simp add: True m_{sb}′ $Write_{sb}$ ts_{sb}′ $\mathcal{O}_{sb}$′ volatile reads-consistent-pending-write-antimono)
next
case False
from j-bound **have** j-bound′: j < length ts_{sb}
by (simp add: ts_{sb}′)
moreover from ts-j False **have** ts-j′: ts_{sb} ! j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
using j-bound **by** (simp add: ts_{sb}′)
ultimately have consis-m: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j
by (rule valid-reads)
from a-unowned-by-others [rule-format, OF j-bound′ False] ts-j′
have a-unowned:a ∉ $\mathcal{O}_j$ ∪ all-acquired sb_j
by simp

let ?take-sb_j = takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j
let ?drop-sb_j = dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j

from a-unowned acquired-reads-all-acquired [of True ?take-sb_j $\mathcal{O}_j$]
all-acquired-append [of ?take-sb_j ?drop-sb_j]
have a-not-acq-reads: a ∉ acquired-reads True ?take-sb_j $\mathcal{O}_j$
by auto
moreover
note a-unfw= a-notin-unforwarded-non-volatile-reads-drop [OF j-bound′ ts-j′ False]
ultimately
show ?thesis
using reads-consistent-mem-eq-on-unforwarded-non-volatile-reads-drop [**where** W={} **and**

A=unforwarded-non-volatile-reads ?drop-sb_j {} ∪ acquired-reads True ?take-sb_j $\mathcal{O}_j$ **and**
m′= (m_{sb}(a:=v)), OF - - - consis-m]
by (fastforce simp add: m_{sb}′)
qed
qed

have valid-own′: valid-ownership $\mathcal{S}_{sb}$′ ts_{sb}′
proof (intro-locales)
show outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}_{sb}$′ ts_{sb}′
proof
fix j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j p_j
assume j-bound: j < length ts_{sb}′
assume ts_{sb}′-j: ts_{sb}′!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$′ $\mathcal{O}_j$ sb_j
proof (cases j=i)
case True
from outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ts_{sb}-i]
have non-volatile-owned-or-read-only False
($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) ($\mathcal{O}_{sb}$ ∪ A − R) sb′
by (auto simp add: sb $Write_{sb}$ volatile non-volatile-owned-or-read-only-pending-write-antimono)
then show ?thesis
using True i-bound ts_{sb}′-j
by (auto simp add: ts_{sb}′ $\mathcal{S}_{sb}$′ sb $\mathcal{O}_{sb}$′)
next
case False
from j-bound **have** j-bound′: j < length ts_{sb}
by (auto simp add: ts_{sb}′)
with ts_{sb}′-j False i-bound
have ts_{sb}-j: ts_{sb}!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (auto simp add: ts_{sb}′)

note nvo = outstanding-non-volatile-refs-owned-or-read-only [OF j-bound′ ts_{sb}-j]

from read-only-unowned [OF i-bound ts_{sb}-i] R-owned
have R ∩ read-only $\mathcal{S}_{sb}$ = {}
by auto
with read-only-reads-unowned [OF j-bound′ i-bound False ts_{sb}-j ts_{sb}-i] L-subset
have ∀ a∈read-only-reads
(acquired True (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j) $\mathcal{O}_j$)
(dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_j).
a ∈ read-only $\mathcal{S}_{sb}$ ⟶ a ∈ read-only ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L)
by (auto simp add: in-read-only-convs sb $Write_{sb}$ volatile)
from non-volatile-owned-or-read-only-read-only-reads-eq′ [OF nvo this]
have non-volatile-owned-or-read-only False ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) $\mathcal{O}_j$ sb_j.
thus ?thesis **by** (simp add: $\mathcal{S}_{sb}$′)
qed
qed
next

show outstanding-volatile-writes-unowned-by-others ts_{sb}'
proof (unfold-locales)
fix i_1 j p_1 is_1 $\mathcal{O}_1$ $\mathcal{R}_1$ $\mathcal{D}_1$ xs_1 sb_1 p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs_j sb_j
assume i_1-bound: i_1 < length ts_{sb}'
assume j-bound: j < length ts_{sb}'
assume i_1-j: $i_1 \neq j$
assume ts-i_1: $ts_{sb}'!i_1 = (p_1,is_1,xs_1,sb_1,\mathcal{D}_1,\mathcal{O}_1,\mathcal{R}_1)$
assume ts-j: $ts_{sb}'!j = (p_j,is_j, xs_j,sb_j,\mathcal{D}_j,\mathcal{O}_j,\mathcal{R}_j)$
show ($\mathcal{O}_j$ ∪ all-acquired sb_j) ∩ outstanding-refs is-volatile-$Write_{sb}$ sb_1 = {}
proof (cases i_1=i)
case True
from i_1-j True **have** neq-i-j: i≠j
by auto
from j-bound **have** j-bound′: j < length ts_{sb}
by (simp add: ts_{sb}')
from ts-j neq-i-j **have** ts-j′: $ts_{sb}!j = (p_j,is_j,xs_j,sb_j,\mathcal{D}_j,\mathcal{O}_j,\mathcal{R}_j)$
by (simp add: ts_{sb}')
from outstanding-volatile-writes-unowned-by-others [OF i-bound j-bound′ neq-i-j
ts_{sb}-i ts-j′] ts-i_1 i-bound ts_{sb}-i True **show** ?thesis
by (clarsimp simp add: ts_{sb}' sb $Write_{sb}$ volatile)
next
case False
note i_1-i = this
from i_1-bound **have** i_1-bound′: i_1 < length ts_{sb}
by (simp add: ts_{sb}' sb)
hence i_1-bound″: i_1 < length (map owned ts_{sb})
by auto
from ts-i_1 False **have** ts-i_1′: $ts_{sb}!i_1 = (p_1,is_1,xs_1,sb_1,\mathcal{D}_1,\mathcal{O}_1,\mathcal{R}_1)$
by (simp add: ts_{sb}' sb)
show ?thesis
proof (cases j=i)
case True
from outstanding-volatile-writes-unowned-by-others [OF i_1-bound′ i-bound i_1-i ts-i_1′
ts_{sb}-i]
have ($\mathcal{O}_{sb}$ ∪ all-acquired sb) ∩ outstanding-refs is-volatile-$Write_{sb}$ sb_1 = {}.
then show ?thesis
using True i_1-i ts-j ts_{sb}-i i-bound
by (auto simp add: sb $Write_{sb}$ volatile ts_{sb}' $\mathcal{O}_{sb}'$)
next
case False
from j-bound **have** j-bound′: j < length ts_{sb}
by (simp add: ts_{sb}')
from ts-j False **have** ts-j′: $ts_{sb}!j = (p_j,is_j,xs_j,sb_j,\mathcal{D}_j,\mathcal{O}_j,\mathcal{R}_j)$
by (simp add: ts_{sb}')
from outstanding-volatile-writes-unowned-by-others
[OF i_1-bound′ j-bound′ i_1-j ts-i_1′ ts-j′]
show ($\mathcal{O}_j$ ∪ all-acquired sb_j) ∩ outstanding-refs is-volatile-$Write_{sb}$ sb_1 = {} .
qed
qed
qed

next
show read-only-reads-unowned ts_{sb}'
proof
fix n m
fix p_n is_n $\mathcal{O}_n$ $\mathcal{R}_n$ $\mathcal{D}_n$ j_n sb_n p_m is_m $\mathcal{O}_m$ $\mathcal{R}_m$ $\mathcal{D}_m$ j_m sb_m
assume n-bound: n < length ts_{sb}'
and m-bound: m < length ts_{sb}'
and neq-n-m: n≠m
and nth: ts_{sb}'!n = (p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
and mth: ts_{sb}'!m =(p_m, is_m, j_m, sb_m, $\mathcal{D}_m$, $\mathcal{O}_m$,$\mathcal{R}_m$)
from n-bound **have** n-bound′: n < length ts_{sb} **by** (simp add: ts_{sb}')
from m-bound **have** m-bound′: m < length ts_{sb} **by** (simp add: ts_{sb}')
show ($\mathcal{O}_m$ ∪ all-acquired sb_m) ∩
read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n) $\mathcal{O}_n$)
(dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n) =
{}
proof (cases m=i)
case True
with neq-n-m **have** neq-n-i: n≠i
by auto

with n-bound nth i-bound **have** nth′: ts_{sb}!n =(p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
by (auto simp add: ts_{sb}')
note read-only-reads-unowned [OF n-bound′ i-bound neq-n-i nth′ ts_{sb}-i]
then
show ?thesis
using True ts_{sb}-i neq-n-i nth mth n-bound′ m-bound′ L-subset
by (auto simp add: ts_{sb}' $\mathcal{O}_{sb}'$ sb $Write_{sb}$ volatile)
next
case False
note neq-m-i = this
with m-bound mth i-bound **have** mth′: ts_{sb}!m = (p_m, is_m, j_m, sb_m, $\mathcal{D}_m$, $\mathcal{O}_m$,$\mathcal{R}_m$)
by (auto simp add: ts_{sb}')
show ?thesis
proof (cases n=i)
case True
from read-only-reads-append [of ($\mathcal{O}_{sb}$ ∪ A − R) (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n)
(dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n)]
have read-only-reads
(acquired True (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n) ($\mathcal{O}_{sb}$ ∪ A − R))
(dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n) ⊆ read-only-reads ($\mathcal{O}_{sb}$ ∪ A − R) sb_n
by auto

with ts_{sb}-i nth mth neq-m-i n-bound′ True
read-only-reads-unowned [OF i-bound m-bound′ False [symmetric] ts_{sb}-i mth′]
show ?thesis
by (auto simp add: ts_{sb}' sb $\mathcal{O}_{sb}'$ $Write_{sb}$ volatile)
next

case False
with n-bound nth i-bound **have** nth′: $ts_{sb}!n = (p_n, is_n, j_n, sb_n, \mathcal{D}_n, \mathcal{O}_n, \mathcal{R}_n)$
by (auto simp add: ts_{sb}')
from read-only-reads-unowned [OF n-bound′ m-bound′ neq-n-m nth′ mth′] False neq-m-i
show ?thesis
by (clarsimp)
qed
qed
qed
next
show ownership-distinct ts_{sb}'
proof (unfold-locales)
fix i_1 j p_1 is_1 $\mathcal{O}_1$ $\mathcal{R}_1$ $\mathcal{D}_1$ xs_1 sb_1 p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs_j sb_j
assume i_1-bound: $i_1 <$ length ts_{sb}'
assume j-bound: $j <$ length ts_{sb}'
assume i_1-j: $i_1 \neq j$
assume ts-i_1: $ts_{sb}'!i_1 = (p_1,is_1,xs_1,sb_1,\mathcal{D}_1,\mathcal{O}_1,\mathcal{R}_1)$
assume ts-j: $ts_{sb}'!j = (p_j,is_j,xs_j,sb_j,\mathcal{D}_j,\mathcal{O}_j,\mathcal{R}_j)$
show $(\mathcal{O}_1 \cup$ all-acquired $sb_1) \cap (\mathcal{O}_j \cup$ all-acquired $sb_j)= \{\}$
proof (cases i_1=i)
case True
with i_1-j **have** i-j: $i \neq j$
by simp

from j-bound **have** j-bound′: $j <$ length ts_{sb}
by (simp add: ts_{sb}')
hence j-bound″: $j <$ length (map owned ts_{sb})
by simp
from ts-j i-j **have** ts-j′: $ts_{sb}!j = (p_j,is_j,xs_j,sb_j,\mathcal{D}_j,\mathcal{O}_j,\mathcal{R}_j)$
by (simp add: ts_{sb}')

from ownership-distinct [OF i-bound j-bound′ i-j ts_{sb}-i ts-j′]
show ?thesis
using ts_{sb}-i True ts-i_1 i-bound $\mathcal{O}_{sb}'$
by (auto simp add: ts_{sb}' sb $Write_{sb}$ volatile)
next
case False
note i_1-i = this
from i_1-bound **have** i_1-bound′: $i_1 <$ length ts_{sb}
by (simp add: ts_{sb}')
hence i_1-bound″: $i_1 <$ length (map owned ts_{sb})
by simp
from ts-i_1 False **have** ts-i_1′: $ts_{sb}!i_1 = (p_1,is_1,xs_1,sb_1,\mathcal{D}_1,\mathcal{O}_1,\mathcal{R}_1)$
by (simp add: ts_{sb}')
show ?thesis
proof (cases j=i)
case True
from ownership-distinct [OF i_1-bound′ i-bound i_1-i ts-i_1′ ts_{sb}-i]
show ?thesis
using ts_{sb}-i True ts-j i-bound $\mathcal{O}_{sb}'$

by (auto simp add: ts_{sb}' sb $Write_{sb}$ volatile)
next
case False
from j-bound **have** j-bound′: j < length ts_{sb}
by (simp add: ts_{sb}')
from ts-j False **have** ts-j′: ts_{sb}!j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (simp add: ts_{sb}')
from ownership-distinct [OF i_1-bound′ j-bound′ i_1-j ts-i_1′ ts-j′]
show ?thesis **.**
qed
qed
qed
qed

have valid-sharing′: valid-sharing ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) ts_{sb}'
proof (intro-locales)
show outstanding-non-volatile-writes-unshared ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) ts_{sb}'
proof (unfold-locales)
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ acq_j xs_j sb_j
assume j-bound: j < length ts_{sb}'
assume jth: ts_{sb}' ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show non-volatile-writes-unshared ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) sb_j
proof (cases i=j)
case True
with outstanding-non-volatile-writes-unshared [OF i-bound ts_{sb}-i]
i-bound jth ts_{sb}-i **show** ?thesis
by (clarsimp simp add: ts_{sb}' sb $Write_{sb}$ volatile)
next
case False
from j-bound **have** j-bound′: j < length ts_{sb}
by (auto simp add: ts_{sb}')
from jth False **have** jth′: ts_{sb} ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (auto simp add: ts_{sb}')
from outstanding-non-volatile-writes-unshared [OF j-bound′ jth′]
have unshared: non-volatile-writes-unshared $\mathcal{S}_{sb}$ sb_j**.**

have ∀a∈dom ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) − dom $\mathcal{S}_{sb}$. a ∉ outstanding-refs is-non-volatile-$Write_{sb}$ sb_j
proof −
{
fix a
assume a-in: a ∈ dom ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) − dom $\mathcal{S}_{sb}$
hence a-R: a ∈ R
by clarsimp
assume a-in-j: a ∈ outstanding-refs is-non-volatile-$Write_{sb}$ sb_j
have False
proof −
from non-volatile-owned-or-read-only-outstanding-non-volatile-writes [OF
outstanding-non-volatile-refs-owned-or-read-only [OF j-bound′ jth′]]
a-in-j

have a ∈ $\mathcal{O}_j$ ∪ all-acquired sb_j
by auto

moreover
with ownership-distinct [OF i-bound j-bound′ False ts_{sb}-i jth′] a-R R-owned
show False
by blast
qed
}
thus ?thesis **by** blast
qed

from non-volatile-writes-unshared-no-outstanding-non-volatile-$Write_{sb}$
[OF unshared this]
show ?thesis **.**
qed
qed
next
show sharing-consis ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) ts_{sb}′
proof (unfold-locales)
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs_j sb_j
assume j-bound: j < length ts_{sb}′
assume jth: ts_{sb}′ ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show sharing-consistent ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) $\mathcal{O}_j$ sb_j
proof (cases i=j)
case True
with i-bound jth ts_{sb}-i sharing-consis [OF i-bound ts_{sb}-i]
show ?thesis
by (clarsimp simp add: ts_{sb}′ sb $Write_{sb}$ volatile $\mathcal{O}_{sb}$′)
next
case False
from j-bound **have** j-bound′: j < length ts_{sb}
by (auto simp add: ts_{sb}′)
from jth False **have** jth′: ts_{sb} ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (auto simp add: ts_{sb}′)
from sharing-consis [OF j-bound′ jth′]
have consis: sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_j$ sb_j**.**

have acq-cond: all-acquired sb_j ∩ dom $\mathcal{S}_{sb}$ − dom ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) = {}
proof −
{
fix a
assume a-acq: a ∈ all-acquired sb_j
assume a ∈ dom $\mathcal{S}_{sb}$
assume a-L: a ∈ L
have False
proof −
from ownership-distinct [OF i-bound j-bound′ False ts_{sb}-i jth′]

have A ∩ all-acquired sb_j = {}
  by (auto simp add: sb $Write_{sb}$ volatile)
with a-acq a-L L-subset
show False
  by blast
qed
}
thus ?thesis
  by auto
qed
have uns-cond: all-unshared sb_j ∩ dom ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) − dom $\mathcal{S}_{sb}$ = {}
proof −
{
fix a
assume a-uns: a ∈ all-unshared sb_j
assume a ∉ L
assume a-R: a ∈ R
have False
proof −
from unshared-acquired-or-owned [OF consis] a-uns
have a ∈ all-acquired sb_j ∪ $\mathcal{O}_j$ **by** auto
with ownership-distinct [OF i-bound j-bound′ False ts_{sb}-i jth′] R-owned a-R
show False
  by blast
qed
}
thus ?thesis
  by auto
qed

from sharing-consistent-preservation [OF consis acq-cond uns-cond]
show ?thesis
by (simp add: ts_{sb}′)
qed
qed
next
show read-only-unowned ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) ts_{sb}′
proof
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs_j sb_j
assume j-bound: j < length ts_{sb}′
assume jth: ts_{sb}′ ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show $\mathcal{O}_j$ ∩ read-only ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) = {}
proof (cases i=j)
case True
from read-only-unowned [OF i-bound ts_{sb}-i] R-owned A-R
have ($\mathcal{O}_{sb}$ ∪ A − R) ∩ read-only ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) = {}
by (auto simp add: in-read-only-convs)
with jth ts_{sb}-i i-bound True
show ?thesis
by (auto simp add: $\mathcal{O}_{sb}$′ ts_{sb}′)

next
case False
from j-bound **have** j-bound′: j < length ts_{sb}
by (auto simp add: ts_{sb}′)
with False jth **have** jth′: ts_{sb} ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (auto simp add: ts_{sb}′)
from read-only-unowned [OF j-bound′ jth′]
have $\mathcal{O}_j \cap$ read-only $\mathcal{S}_{sb}$ = {}.
moreover
from ownership-distinct [OF i-bound j-bound′ False ts_{sb}-i jth′] R-owned
have ($\mathcal{O}_{sb} \cup$ A) $\cap \mathcal{O}_j$ = {}
by (auto simp add: sb $Write_{sb}$ volatile)
moreover note R-owned A-R
ultimately show ?thesis
by (fastforce simp add: in-read-only-convs split: if-split-asm)
qed
qed
next
show unowned-shared ($\mathcal{S}_{sb} \oplus_W R \ominus_A L$) ts_{sb}′
proof (unfold-locales)
show $- \bigcup((\lambda(-,-, -, -,-, \mathcal{O},-).\ \mathcal{O})$ ‘ set ts_{sb}′) $\subseteq$ dom ($\mathcal{S}_{sb} \oplus_W R \ominus_A L$)
proof −

have s: $\bigcup((\lambda(-,-, -, -,-, \mathcal{O},-).\ \mathcal{O})$ ‘ set ts_{sb}′) =
$\bigcup((\lambda(-,-, -, -,-, \mathcal{O},-).\ \mathcal{O})$ ‘ set ts_{sb}) $\cup$ A − R

apply (unfold ts_{sb}′ $\mathcal{O}_{sb}$′)
apply (rule acquire-release-ownership-nth-update [OF R-owned i-bound ts_{sb}-i])
apply (rule local.ownership-distinct-axioms)
done

note unowned-shared L-subset A-R
then
show ?thesis
apply (simp only: s)
apply auto
done
qed
qed
next
show no-outstanding-write-to-read-only-memory ($\mathcal{S}_{sb} \oplus_W R \ominus_A L$) ts_{sb}′
proof
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ acq_j xs_j sb_j
assume j-bound: j < length ts_{sb}′
assume jth: ts_{sb}′ ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show no-write-to-read-only-memory ($\mathcal{S}_{sb} \oplus_W R \ominus_A L$) sb_j
proof (cases i=j)
case True
with jth ts_{sb}-i i-bound no-outstanding-write-to-read-only-memory [OF i-bound ts_{sb}-i]
show ?thesis

by (auto simp add: sb ts_{sb}' $Write_{sb}$ volatile)
next
case False
from j-bound **have** j-bound′: j < length ts_{sb}
by (auto simp add: ts_{sb}')
with False jth **have** jth′: ts_{sb} ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (auto simp add: ts_{sb}')
from no-outstanding-write-to-read-only-memory [OF j-bound′ jth′]
have nw: no-write-to-read-only-memory $\mathcal{S}_{sb}$ sb_j**.**
have R ∩ outstanding-refs is-$Write_{sb}$ sb_j = {}
proof −
note dist = ownership-distinct [OF i-bound j-bound′ False ts_{sb}-i jth′]
from non-volatile-owned-or-read-only-outstanding-non-volatile-writes
[OF outstanding-non-volatile-refs-owned-or-read-only [OF j-bound′ jth′]]
dist
have outstanding-refs is-non-volatile-$Write_{sb}$ sb_j ∩ $\mathcal{O}_{sb}$ = {}
by auto
moreover
from outstanding-volatile-writes-unowned-by-others [OF j-bound′ i-bound
False [symmetric] jth′ ts_{sb}-i]
have outstanding-refs is-volatile-$Write_{sb}$ sb_j ∩ $\mathcal{O}_{sb}$ = {}
by auto
ultimately have outstanding-refs is-$Write_{sb}$ sb_j ∩ $\mathcal{O}_{sb}$ = {}
by (auto simp add: misc-outstanding-refs-convs)
with R-owned
show ?thesis **by** blast
qed
then
have ∀a∈outstanding-refs is-$Write_{sb}$ sb_j.
a ∈ read-only ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) ⟶ a ∈ read-only $\mathcal{S}_{sb}$
by (auto simp add: in-read-only-convs)

from no-write-to-read-only-memory-read-only-reads-eq [OF nw this]
show ?thesis **.**
qed
qed
qed

from direct-memop-step.WriteVolatile [OF]
have (Write True a (D, f) A L R W# is′,
?j, (), m,$\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$, release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$,$\mathcal{S}$) →
(is′, ?j, (), m (a := v),True, acquired True ?take-sb $\mathcal{O}_{sb}$ ∪ A − R, Map.empty,$\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)
by (simp add: f-v′ [symmetric])

from direct-computation.Memop [OF i-bound′ ts-i this]
have store-step:
(ts, m, $\mathcal{S}$) $\Rightarrow_d$ (?ts′, m(a := v),$\mathcal{S}$ $\oplus_W$ R $\ominus_A$ L)**.**

have sb′-split:

$sb' = \text{takeWhile} (\text{Not} \circ \text{is-volatile-Write}_{sb})\ sb' @$
$\quad\quad \text{dropWhile} (\text{Not} \circ \text{is-volatile-Write}_{sb})\ sb'$
by simp

from reads-consis
have no-vol-reads: outstanding-refs is-volatile-Read$_{sb}$ sb′ = {}
by (simp add: sb Write$_{sb}$ True)
hence outstanding-refs is-volatile-Read$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′)
= {}
by (auto simp add: outstanding-refs-conv dest: set-takeWhileD)
moreover
have outstanding-refs is-volatile-Write$_{sb}$
(takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′) = {}
proof −
have ∀ r ∈ set (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′). ¬ (is-volatile-Write$_{sb}$ r)
by (auto dest: set-takeWhileD)
thus ?thesis
by (simp add: outstanding-refs-conv)
qed
ultimately
have no-volatile:
outstanding-refs is-volatile (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′) = {}
by (auto simp add: outstanding-refs-conv is-volatile-split)

moreover

from no-vol-reads **have** ∀ r ∈ set sb′. ¬ is-volatile-Read$_{sb}$ r
by (fastforce simp add: outstanding-refs-conv is-volatile-Read$_{sb}$-def
split: memref.splits)
hence ∀ r ∈ set sb′. (Not ∘ is-volatile-Write$_{sb}$) r = (Not ∘ is-volatile) r
by (auto simp add: is-volatile-split)

hence takeWhile-eq: (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′) =
(takeWhile (Not ∘ is-volatile) sb′)
apply −
apply (rule takeWhile-cong)
apply auto
done

from leq
have leq′: length ts$_{sb}$ = length ?ts′
by simp
hence i-bound-ts′: i < length ?ts′ **using** i-bound **by** simp

from is′-sim
have is′-sim-split:
instrs
(takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′ @
dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb′) @ is$_{sb}$ =
is′ @ prog-instrs (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′ @

dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb′)
by (simp add: sb′-split [symmetric])

from reads-consistent-flush-all-until-volatile-write [OF ‹valid-ownership-and-sharing $\mathcal{S}_{sb}$ ts$_{sb}$›
i-bound ts$_{sb}$-i reads-consis]
have reads-consistent True (acquired True ?take-sb $\mathcal{O}_{sb}$) m (Write$_{sb}$ True a (D,f) v A L R W#sb′)
by (simp add: m sb Write$_{sb}$ volatile)

hence reads-consistent True (acquired True ?take-sb $\mathcal{O}_{sb}$ ∪ A − R) (m(a:=v)) sb′
by simp
from reads-consistent-takeWhile [OF this]
have r-consis′: reads-consistent True (acquired True ?take-sb $\mathcal{O}_{sb}$ ∪ A − R) (m(a:=v))
(takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′).

from last-prog **have** last-prog-sb′: last-prog p$_{sb}$ sb′ = p$_{sb}$
by (simp add: sb Write$_{sb}$)

from valid-write-sops [OF i-bound ts$_{sb}$-i]
have ∀sop ∈ write-sops sb′. valid-sop sop
by (auto simp add: sb Write$_{sb}$)
hence valid-sop′: ∀sop∈write-sops (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′).
valid-sop sop
by (fastforce dest: set-takeWhileD simp add: in-write-sops-conv)

from no-volatile
have no-volatile-Read$_{sb}$:
outstanding-refs is-volatile-Read$_{sb}$ (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′) =
{}
by (auto simp add: outstanding-refs-conv is-volatile-Read$_{sb}$-def split: memref.splits)
from flush-store-buffer-append [OF i-bound-ts′ is′-sim-split, simplified,
OF causal-program-history-sb′ ts′-i refl last-prog-sb′ r-consis′ hist-consis′
valid-sop′ dist-sb′ no-volatile-Read$_{sb}$-volatile-reads-consistent [OF no-volatile-Read$_{sb}$],
where $\mathcal{S}$=($\mathcal{S}$ ⊕$_{W}$ R ⊖$_{A}$ L)]

obtain is″ **where**
is″-sim: instrs (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb′) @ is$_{sb}$ =
is″ @ prog-instrs (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb′) **and**

steps: (?ts′, m(a := v), $\mathcal{S}$ ⊕$_{W}$ R ⊖$_{A}$ L) ⇒$_{d}$*
(ts[i := (last-prog (hd-prog p$_{sb}$ (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb′))
(takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′),
is″,
j$_{sb}$ |‘ (dom j$_{sb}$ −
read-tmps (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb′)),

(), True, acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′)
(acquired True ?take-sb $\mathcal{O}_{sb}$ ∪ A − R),
release (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′)
(dom ($\mathcal{S}$ ⊕$_W$ R ⊖$_A$ L)) Map.empty)],
flush (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′) (m(a := v)),
share (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′) ($\mathcal{S}$ ⊕$_W$ R ⊖$_A$ L))

by (auto)

note sim-flush = r-rtranclp-rtranclp [OF store-step steps]

moreover
note flush-commute =
flush-flush-all-until-volatile-write-Write$_{sb}$-volatile-commute [OF i-bound ts$_{sb}$-i [simplified sb Write$_{sb}$ True]
outstanding-refs-is-Write$_{sb}$-takeWhile-disj a-notin-others′]

from last-prog-hd-prog-append′ [**where** sb=(takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′)
and sb′=(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb′),
simplified sb′-split [symmetric], OF hist-consis′ last-prog-sb′]
have last-prog-eq:
last-prog (hd-prog p$_{sb}$ (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb′))
(takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′) =
hd-prog p$_{sb}$ (dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb′).

have take-emtpy: takeWhile (Not ∘ is-volatile-Write$_{sb}$) (r#sb) = []
by (simp add: Write$_{sb}$ True)

have dist-sb′: ∀i p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
i < length ts$_{sb}$ ⟶
ts$_{sb}$! i = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$) ⟶
(all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∪
all-unshared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) ∪
all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb)) ∩
(all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′) ∪
all-unshared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′) ∪
all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′)) =
{}
proof −
{
fix j p$_j$ is$_j$ $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j$_j$ sb$_j$ x
assume j-bound: j < length ts$_{sb}$
assume jth: ts$_{sb}$!j = (p$_j$,is$_j$, j$_j$,sb$_j$,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
assume x-shared: x ∈ all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$) ∪
all-unshared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$) ∪
all-acquired (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb$_j$)
assume x-sb′: x ∈ (all-shared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′) ∪
all-unshared (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′) ∪

all-acquired (takeWhile (Not ∘ is-volatile-$\mathsf{Write_{sb}}$) sb′))
have False
proof (cases i=j)
case True **with** x-shared ts_{sb}-i jth **show** False **by** (simp add: sb volatile $\mathsf{Write_{sb}}$)
next
case False
from x-shared all-shared-acquired-or-owned [OF sharing-consis [OF j-bound jth]]
unshared-acquired-or-owned [OF sharing-consis [OF j-bound jth]]
all-shared-append [of (takeWhile (Not ∘ is-volatile-$\mathsf{Write_{sb}}$) sb_j)
(dropWhile (Not ∘ is-volatile-$\mathsf{Write_{sb}}$) sb_j)]
all-unshared-append [of (takeWhile (Not ∘ is-volatile-$\mathsf{Write_{sb}}$) sb_j)
(dropWhile (Not ∘ is-volatile-$\mathsf{Write_{sb}}$) sb_j)]
all-acquired-append [of (takeWhile (Not ∘ is-volatile-$\mathsf{Write_{sb}}$) sb_j)
(dropWhile (Not ∘ is-volatile-$\mathsf{Write_{sb}}$) sb_j)]
have x $\in$ all-acquired $sb_j \cup \mathcal{O}_j$
by auto
moreover
from x-sb′ all-shared-acquired-or-owned [OF sharing-consis [OF i-bound ts_{sb}-i]]
unshared-acquired-or-owned [OF sharing-consis [OF i-bound ts_{sb}-i]]
all-shared-append [of (takeWhile (Not ∘ is-volatile-$\mathsf{Write_{sb}}$) sb′)
(dropWhile (Not ∘ is-volatile-$\mathsf{Write_{sb}}$) sb′)]
all-unshared-append [of (takeWhile (Not ∘ is-volatile-$\mathsf{Write_{sb}}$) sb′)
(dropWhile (Not ∘ is-volatile-$\mathsf{Write_{sb}}$) sb′)]
all-acquired-append [of (takeWhile (Not ∘ is-volatile-$\mathsf{Write_{sb}}$) sb′)
(dropWhile (Not ∘ is-volatile-$\mathsf{Write_{sb}}$) sb′)]
have x $\in$ all-acquired sb $\cup \mathcal{O}_{sb}$
by (auto simp add: sb $\mathsf{Write_{sb}}$ volatile)
moreover
note ownership-distinct [OF i-bound j-bound False ts_{sb}-i jth]
ultimately show False **by** blast
qed
}
thus ?thesis **by** blast
qed

have dist-R-L-A: $\forall$j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
j < length ts_{sb} $\longrightarrow$ i$\neq$ j$\longrightarrow$
ts_{sb} ! j = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$) $\longrightarrow$
(all-shared sb $\cup$ all-unshared sb $\cup$ all-acquired sb) $\cap$ (R $\cup$ L $\cup$ A) = {}
proof –
{
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j x
assume j-bound: j < length ts_{sb}
assume neq-i-j: i $\neq$ j
assume jth: ts_{sb}!j = (p_j,is_j, j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
assume x-shared: x $\in$ all-shared sb_j $\cup$
all-unshared sb_j $\cup$
all-acquired sb_j
assume x-R-L-A: x $\in$ R $\cup$ L $\cup$ A

have False
proof –
from x-shared all-shared-acquired-or-owned [OF sharing-consis [OF j-bound jth]]
unshared-acquired-or-owned [OF sharing-consis [OF j-bound jth]]

have x ∈ all-acquired sb_j ∪ $\mathcal{O}_j$
by auto
moreover
from x-R-L-A R-owned L-subset
have x ∈ all-acquired sb ∪ $\mathcal{O}_{sb}$
by (auto simp add: sb $Write_{sb}$ volatile)
moreover
note ownership-distinct [OF i-bound j-bound neq-i-j ts_{sb}-i jth]
ultimately show False **by** blast
qed
}
thus ?thesis **by** blast
qed
from local.ownership-distinct-axioms **have** ownership-distinct ts_{sb} **.**
from local.sharing-consis-axioms **have** sharing-consis $\mathcal{S}_{sb}$ ts_{sb}**.**
note share-commute=
share-all-until-volatile-write-flush-commute [OF take-empty ‹ownership-distinct ts_{sb}› ‹sharing-consis $\mathcal{S}_{sb}$ ts_{sb}› i-bound ts_{sb}-i dist-sb′ dist-R-L-A]

have rel-commute-empty:
release (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb′) (dom $\mathcal{S}$ ∪ R – L) Map.empty =
release (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb′) (dom $\mathcal{S}_{sb}$ ∪ R – L) Map.empty
proof –
{
fix a
assume a-in: a ∈ all-shared (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb′)
have (a ∈ (dom $\mathcal{S}$ ∪ R – L)) = (a ∈ (dom $\mathcal{S}_{sb}$ ∪ R – L))
proof –

from all-shared-acquired-or-owned [OF sharing-consis [OF i-bound ts_{sb}-i]] a-in
all-shared-append [of (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb′) (dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb′)]
have a ∈ $\mathcal{O}_{sb}$ ∪ all-acquired sb
by (auto simp add: sb $Write_{sb}$ volatile)
from share-all-until-volatile-write-thread-local [OF ‹ownership-distinct ts_{sb}› ‹sharing-consis $\mathcal{S}_{sb}$ ts_{sb}› i-bound ts_{sb}-i this]
have $\mathcal{S}$ a = $\mathcal{S}_{sb}$ a
by (auto simp add: sb $Write_{sb}$ volatile $\mathcal{S}$)
then show ?thesis
by (auto simp add: domIff)
qed
}
then show ?thesis

```
      apply −
      apply (rule release-all-shared-exchange)
      apply auto
      done
    qed

    {
fix j pj isj 𝒪j ℛj 𝒟j jj sbj x
assume jth: tssb!j = (pj,isj,jj,sbj,𝒟j,𝒪j,ℛj)
assume j-bound: j < length tssb
     assume neq: i ≠ j
     have release (takeWhile (Not ∘ is-volatile-Writesb) sbj)
                  (dom 𝒮sb ∪ R − L) ℛj
        = release (takeWhile (Not ∘ is-volatile-Writesb) sbj)
                  (dom 𝒮sb) ℛj
     proof −
      {
       fix a
       assume a-in: a ∈ all-shared (takeWhile (Not ∘ is-volatile-Writesb) sbj)
       have (a ∈ (dom 𝒮sb ∪ R − L)) = (a ∈ dom 𝒮sb)
       proof −
        from ownership-distinct [OF i-bound j-bound neq  tssb-i jth]

        have A-dist: A ∩ (𝒪j ∪ all-acquired sbj) = {}
         by (auto simp add: sb Writesb volatile)

        from  all-shared-acquired-or-owned [OF sharing-consis [OF j-bound jth]] a-in
          all-shared-append [of (takeWhile (Not ∘ is-volatile-Writesb) sbj)
          (dropWhile (Not ∘ is-volatile-Writesb) sbj)]
        have a-in: a ∈ 𝒪j ∪ all-acquired sbj
         by auto
        with ownership-distinct [OF i-bound j-bound neq  tssb-i jth]
        have a ∉ (𝒪sb ∪ all-acquired sb) by auto


        with A-dist R-owned A-R A-shared-owned L-subset a-in
        obtain a ∉ R and a ∉ L
         by fastforce
        then show ?thesis by auto
       qed
      }
      then
      show ?thesis
       apply −
       apply (rule release-all-shared-exchange)
       apply auto
       done
     qed
    }
    note release-commute = this
```

have (ts$_{\mathsf{sb}}$ [i := (p$_{\mathsf{sb}}$,is$_{\mathsf{sb}}$, j$_{\mathsf{sb}}$, sb$'$, $\mathcal{D}_{\mathsf{sb}}$, $\mathcal{O}_{\mathsf{sb}}$ $\cup$ A $-$ R,Map.empty)],m$_{\mathsf{sb}}$(a:=v),$\mathcal{S}_{\mathsf{sb}}{}'$) $\sim$
(ts[i := (last-prog (hd-prog p$_{\mathsf{sb}}$ (dropWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$'$))
(takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$'$),
is$''$,
j$_{\mathsf{sb}}$ |' (dom j$_{\mathsf{sb}}$ $-$
read-tmps (dropWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$'$)),
(),True, acquired True (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$'$)
(acquired True ?take-sb $\mathcal{O}_{\mathsf{sb}}$ $\cup$ A $-$ R),
release (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$'$)
(dom ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)) Map.empty)],
flush (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$'$) (m(a := v)),
share (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$'$) ($\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L))
apply (rule sim-config.intros)
apply (simp add: flush-commute m)
apply (clarsimp simp add: $\mathcal{S}_{\mathsf{sb}}{}'$ $\mathcal{S}$ share-commute simp del: restrict-restrict)
using leq
apply simp
using i-bound i-bound$'$ ts-sim $\mathcal{D}$
apply (clarsimp simp add: Let-def nth-list-update is$''$-sim last-prog-eq sb Write$_{\mathsf{sb}}$ volatile $\mathcal{S}_{\mathsf{sb}}{}'$
rel-commute-empty
split: if-split-asm)
apply (rule conjI)
apply blast
apply clarsimp
apply (frule (2) release-commute)
apply clarsimp
apply fastforce
done

ultimately
show ?thesis
using valid-own$'$ valid-hist$'$ valid-reads$'$ valid-sharing$'$ tmps-distinct$'$
valid-dd$'$ valid-sops$'$ load-tmps-fresh$'$ enough-flushs$'$
valid-program-history$'$ valid$'$
m$_{\mathsf{sb}}{}'$ $\mathcal{S}_{\mathsf{sb}}{}'$ ts$_{\mathsf{sb}}{}'$
by (auto simp del: fun-upd-apply simp add: $\mathcal{O}_{\mathsf{sb}}{}'$ $\mathcal{R}_{\mathsf{sb}}{}'$)

next

case False
note non-vol = this

from flush Write$_{\mathsf{sb}}$ False
obtain
$\mathcal{O}_{\mathsf{sb}}{}'$: $\mathcal{O}_{\mathsf{sb}}{}'$=$\mathcal{O}_{\mathsf{sb}}$ **and**
$\mathcal{S}_{\mathsf{sb}}{}'$: $\mathcal{S}_{\mathsf{sb}}{}'$=$\mathcal{S}_{\mathsf{sb}}$ **and**
$\mathcal{R}_{\mathsf{sb}}{}'$: $\mathcal{R}_{\mathsf{sb}}{}'$ = $\mathcal{R}_{\mathsf{sb}}$
by cases (auto simp add: sb)

from non-volatile-owned non-vol **have** a-owned: $a \in \mathcal{O}_{sb}$
by simp

{
fix j
fix p_j is_{sbj} $\mathcal{O}_j$ $\mathcal{D}_{sbj}$ j_j $\mathcal{R}_j$ sb_j
assume j-bound: $j <$ length ts_{sb}
assume ts_{sb}-j: $ts_{sb}!j=(p_j,is_{sbj},j_j,sb_j,\mathcal{D}_{sbj},\mathcal{O}_j,\mathcal{R}_j)$
assume neq-i-j: $i \neq j$
have $a \notin$ unforwarded-non-volatile-reads (dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j) {}
proof
let ?take-sb_j = takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j
let ?drop-sb_j = dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb_j
assume a-in: $a \in$ unforwarded-non-volatile-reads ?drop-sb_j {}

from a-unowned-by-others [rule-format, OF j-bound neq-i-j] ts_{sb}-j
obtain a-unowned: $a \notin \mathcal{O}_j$ **and** a-unacq: $a \notin$ all-acquired sb_j
by auto
with all-acquired-append [of ?take-sb_j ?drop-sb_j] acquired-takeWhile-non-volatile-Write$_{sb}$ [of sb_j $\mathcal{O}_j$]
have a-unacq-take: $a \notin$ acquired True ?take-sb_j $\mathcal{O}_j$
by (auto)

note nvo-j = outstanding-non-volatile-refs-owned-or-read-only [OF j-bound ts_{sb}-j]

from non-volatile-owned-or-read-only-drop [OF nvo-j]
have nvo-drop-j: non-volatile-owned-or-read-only True (share ?take-sb_j $\mathcal{S}_{sb}$)
(acquired True ?take-sb_j $\mathcal{O}_j$) ?drop-sb_j **.**
from in-unforwarded-non-volatile-reads-non-volatile-Read$_{sb}$ [OF a-in]
have a-in′: $a \in$ outstanding-refs is-non-volatile-Read$_{sb}$?drop-sb_j**.**

from non-volatile-owned-or-read-only-outstanding-refs [OF nvo-drop-j] a-in′
have $a \in$ acquired True ?take-sb_j $\mathcal{O}_j$ $\cup$ all-acquired ?drop-sb_j $\cup$
read-only-reads (acquired True ?take-sb_j $\mathcal{O}_j$) ?drop-sb_j
by (auto simp add: misc-outstanding-refs-convs)

moreover
from acquired-append [of True ?take-sb_j ?drop-sb_j $\mathcal{O}_j$] acquired-all-acquired [of True ?take-sb_j $\mathcal{O}_j$]
all-acquired-append [of ?take-sb_j ?drop-sb_j]
have acquired True ?take-sb_j $\mathcal{O}_j$ $\cup$ all-acquired ?drop-sb_j $\subseteq \mathcal{O}_j \cup$ all-acquired sb_j
by auto
ultimately
have $a \in$ read-only-reads (acquired True ?take-sb_j $\mathcal{O}_j$) ?drop-sb_j
using a-owned ownership-distinct [OF i-bound j-bound neq-i-j ts_{sb}-i ts_{sb}-j]
by auto

with read-only-reads-unowned [OF j-bound i-bound neq-i-j [symmetric] ts_{sb}-j ts_{sb}-i] a-owned
show False
by auto
qed
} **note** a-notin-unforwarded-non-volatile-reads-drop = this

have valid-reads': valid-reads m_{sb}' ts_{sb}'
proof (unfold-locales)
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ j_j sb_j
assume j-bound: j < length ts_{sb}'
assume ts-j: ts_{sb}'!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show reads-consistent False $\mathcal{O}_j$ m_{sb}' sb_j
proof (cases i=j)
case True
from reads-consis ts-j j-bound sb **show** ?thesis
by (clarsimp simp add: True m_{sb}' $Write_{sb}$ ts_{sb}' $\mathcal{O}_{sb}'$ False reads-consistent-pending-write-antimono)
next
case False
from j-bound **have** j-bound': j < length ts_{sb}
by (simp add: ts_{sb}')
moreover from ts-j False **have** ts-j': ts_{sb} ! j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
using j-bound **by** (simp add: ts_{sb}')
ultimately have consis-m: reads-consistent False $\mathcal{O}_j$ m_{sb} sb_j
by (rule valid-reads)
from a-unowned-by-others [rule-format, OF j-bound' False] ts-j'
have a-unowned:a $\notin$ $\mathcal{O}_j$ $\cup$ all-acquired sb_j
by simp

let ?take-sb_j = takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j
let ?drop-sb_j = dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb_j

from a-unowned acquired-reads-all-acquired [of True ?take-sb_j $\mathcal{O}_j$]
all-acquired-append [of ?take-sb_j ?drop-sb_j]
have a-not-acq-reads: a $\notin$ acquired-reads True ?take-sb_j $\mathcal{O}_j$
by auto
moreover

note a-unfw= a-notin-unforwarded-non-volatile-reads-drop [OF j-bound' ts-j' False]
ultimately
show ?thesis
using reads-consistent-mem-eq-on-unforwarded-non-volatile-reads-drop [**where** W={} **and**
A=unforwarded-non-volatile-reads ?drop-sb_j {} $\cup$ acquired-reads True ?take-sb_j $\mathcal{O}_j$ **and**
m'= (m_{sb}(a:=v)), OF - - - consis-m]
by (fastforce simp add: m_{sb}')
qed

qed

have valid-own′: valid-ownership $\mathcal{S}_{\mathsf{sb}}'$ ts_{sb}'
proof (intro-locales)
show outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}_{\mathsf{sb}}'$ ts_{sb}'
proof −
from outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ts_{sb}-i] sb
have non-volatile-owned-or-read-only False $\mathcal{S}_{\mathsf{sb}}$ $\mathcal{O}_{\mathsf{sb}}$ sb′
by (auto simp add: $Write_{\mathsf{sb}}$ False)
from outstanding-non-volatile-refs-owned-or-read-only-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' $Write_{\mathsf{sb}}$ False $\mathcal{O}_{\mathsf{sb}}'$ $\mathcal{S}_{\mathsf{sb}}'$)
qed
next
show outstanding-volatile-writes-unowned-by-others ts_{sb}'
proof −
from sb
have out: outstanding-refs is-volatile-$Write_{\mathsf{sb}}$ sb′ $\subseteq$ outstanding-refs is-volatile-$Write_{\mathsf{sb}}$ sb
by (auto simp add: $Write_{\mathsf{sb}}$ False)
have acq: all-acquired sb′ $\subseteq$ all-acquired sb
by (auto simp add: $Write_{\mathsf{sb}}$ False sb)
from outstanding-volatile-writes-unowned-by-others-store-buffer
[OF i-bound ts_{sb}-i out acq]
show ?thesis **by** (simp add: ts_{sb}' $Write_{\mathsf{sb}}$ False $\mathcal{O}_{\mathsf{sb}}'$)
qed
next
show read-only-reads-unowned ts_{sb}'
proof −
have ro: read-only-reads (acquired True (takeWhile (Not $\circ$ is-volatile-$Write_{\mathsf{sb}}$) sb′) $\mathcal{O}_{\mathsf{sb}}$)
(dropWhile (Not $\circ$ is-volatile-$Write_{\mathsf{sb}}$) sb′)
$\subseteq$ read-only-reads (acquired True (takeWhile (Not $\circ$ is-volatile-$Write_{\mathsf{sb}}$) sb) $\mathcal{O}_{\mathsf{sb}}$)
(dropWhile (Not $\circ$ is-volatile-$Write_{\mathsf{sb}}$) sb)
by (auto simp add: sb $Write_{\mathsf{sb}}$ non-vol)
have $\mathcal{O}_{\mathsf{sb}}$ $\cup$ all-acquired sb′ $\subseteq$ $\mathcal{O}_{\mathsf{sb}}$ $\cup$ all-acquired sb
by (auto simp add: sb $Write_{\mathsf{sb}}$ non-vol)
from read-only-reads-unowned-nth-update [OF i-bound ts_{sb}-i ro this]
show ?thesis
by (simp add: ts_{sb}' sb $\mathcal{O}_{\mathsf{sb}}'$)
qed
next
show ownership-distinct ts_{sb}'
proof −
have acq: all-acquired sb′ $\subseteq$ all-acquired sb
by (auto simp add: $Write_{\mathsf{sb}}$ False sb)
with ownership-distinct-instructions-read-value-store-buffer-independent
[OF i-bound ts_{sb}-i]
show ?thesis **by** (simp add: ts_{sb}' $Write_{\mathsf{sb}}$ False $\mathcal{O}_{\mathsf{sb}}'$)
qed
qed

have valid-sharing′: valid-sharing $\mathcal{S}_{\mathsf{sb}}{}'$ $\mathrm{ts}_{\mathsf{sb}}{}'$
proof (intro-locales)
  from outstanding-non-volatile-writes-unshared [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i]
  have non-volatile-writes-unshared $\mathcal{S}_{\mathsf{sb}}$ sb′
    by (auto simp add: sb $\mathrm{Write}_{\mathsf{sb}}$ False)
  from outstanding-non-volatile-writes-unshared-nth-update [OF i-bound this]
  show outstanding-non-volatile-writes-unshared $\mathcal{S}_{\mathsf{sb}}{}'$ $\mathrm{ts}_{\mathsf{sb}}{}'$
    by (simp add: $\mathrm{ts}_{\mathsf{sb}}{}'$ $\mathcal{S}_{\mathsf{sb}}{}'$)
next
  from sharing-consis [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i]
  have sharing-consistent $\mathcal{S}_{\mathsf{sb}}$ $\mathcal{O}_{\mathsf{sb}}$ sb′
    by (auto simp add: sb $\mathrm{Write}_{\mathsf{sb}}$ False)
  from sharing-consis-nth-update [OF i-bound this]
  show sharing-consis $\mathcal{S}_{\mathsf{sb}}{}'$ $\mathrm{ts}_{\mathsf{sb}}{}'$
    by (simp add: $\mathrm{ts}_{\mathsf{sb}}{}'$ $\mathcal{O}_{\mathsf{sb}}{}'$ $\mathcal{S}_{\mathsf{sb}}{}'$)
next
  from read-only-unowned-nth-update [OF i-bound read-only-unowned [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i]
]
  show read-only-unowned $\mathcal{S}_{\mathsf{sb}}{}'$ $\mathrm{ts}_{\mathsf{sb}}{}'$
    by (simp add: $\mathcal{S}_{\mathsf{sb}}{}'$ $\mathrm{ts}_{\mathsf{sb}}{}'$ $\mathcal{O}_{\mathsf{sb}}{}'$)
next
  from unowned-shared-nth-update [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i subset-refl]
  show unowned-shared $\mathcal{S}_{\mathsf{sb}}{}'$ $\mathrm{ts}_{\mathsf{sb}}{}'$
    by (simp add: $\mathrm{ts}_{\mathsf{sb}}{}'$ $\mathcal{O}_{\mathsf{sb}}{}'$ $\mathcal{S}_{\mathsf{sb}}{}'$)
next
  from no-outstanding-write-to-read-only-memory [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i]
  have no-write-to-read-only-memory $\mathcal{S}_{\mathsf{sb}}$ sb′
    by (auto simp add: sb $\mathrm{Write}_{\mathsf{sb}}$ False)
  from no-outstanding-write-to-read-only-memory-nth-update [OF i-bound this]
  show no-outstanding-write-to-read-only-memory $\mathcal{S}_{\mathsf{sb}}{}'$ $\mathrm{ts}_{\mathsf{sb}}{}'$
    by (simp add: $\mathcal{S}_{\mathsf{sb}}{}'$ $\mathrm{ts}_{\mathsf{sb}}{}'$ sb)
qed

from is-sim
obtain is-sim: instrs (dropWhile (Not ∘ is-volatile-$\mathrm{Write}_{\mathsf{sb}}$) sb′) @ $\mathrm{is}_{\mathsf{sb}}$ =
    is @ prog-instrs (dropWhile (Not ∘ is-volatile-$\mathrm{Write}_{\mathsf{sb}}$) sb′)
  by (simp add: suspends sb $\mathrm{Write}_{\mathsf{sb}}$ False)

have (ts,m,$\mathcal{S}$) $\Rightarrow_{\mathsf{d}}{}^*$ (ts,m,$\mathcal{S}$) **by** blast

moreover

note flush-commute =
  flush-all-until-volatile-write-$\mathrm{Write}_{\mathsf{sb}}$-non-volatile-commute [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i [simplified
sb $\mathrm{Write}_{\mathsf{sb}}$ non-vol]
outstanding-refs-is-$\mathrm{Write}_{\mathsf{sb}}$-takeWhile-disj a-notin-others′]

note share-commute =

share-all-until-volatile-write-update-sb [of sb′ sb, OF - i-bound ts_{sb}-i, simplified sb $Write_{sb}$ False, simplified]
have (ts_{sb} [i := (p_{sb},is_{sb},j_{sb}, sb′, $\mathcal{D}_{sb}$, $\mathcal{O}_{sb}$,$\mathcal{R}_{sb}$)], m_{sb}(a:=v),$\mathcal{S}_{sb}$′) ∼
(ts,m,$\mathcal{S}$)
apply (rule sim-config.intros)
apply (simp add: m flush-commute)
apply (clarsimp simp add: $\mathcal{S}$ $\mathcal{S}_{sb}$′ share-commute)
using leq
apply simp
using i-bound i-bound′ is-sim ts-i ts-sim $\mathcal{D}$
apply (clarsimp simp add: Let-def nth-list-update suspends sb $Write_{sb}$ False $\mathcal{S}_{sb}$′
split: if-split-asm)
done

ultimately
show ?thesis
using valid-own′ valid-hist′ valid-reads′ valid-sharing′ tmps-distinct′ m_{sb}′
valid-dd′ valid-sops′ load-tmps-fresh′ enough-flushs′ valid-program-history′ valid′
ts_{sb}′ $\mathcal{O}_{sb}$′ $\mathcal{S}_{sb}$′ $\mathcal{R}_{sb}$′
by (auto simp del: fun-upd-apply)
qed
next
case ($Read_{sb}$ volatile a t v)
from flush this **obtain** m_{sb}′: m_{sb}′=m_{sb} **and**
$\mathcal{O}_{sb}$′: $\mathcal{O}_{sb}$′=$\mathcal{O}_{sb}$ **and** $\mathcal{S}_{sb}$′: $\mathcal{S}_{sb}$′=$\mathcal{S}_{sb}$ **and**
$\mathcal{R}_{sb}$′: $\mathcal{R}_{sb}$′=$\mathcal{R}_{sb}$
by cases (auto simp add: sb)

have valid-own′: valid-ownership $\mathcal{S}_{sb}$′ ts_{sb}′
proof (intro-locales)
show outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}_{sb}$′ ts_{sb}′
proof −
from outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ts_{sb}-i] sb
have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ sb′
by (auto simp add: $Read_{sb}$)
from outstanding-non-volatile-refs-owned-or-read-only-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}′ $Read_{sb}$ $\mathcal{O}_{sb}$′ $\mathcal{S}_{sb}$′)
qed
next
show outstanding-volatile-writes-unowned-by-others ts_{sb}′
proof −
from sb
have out: outstanding-refs is-volatile-$Write_{sb}$ sb′ ⊆ outstanding-refs is-volatile-$Write_{sb}$ sb
by (auto simp add: $Read_{sb}$)
have acq: all-acquired sb′ ⊆ all-acquired sb
by (auto simp add: $Read_{sb}$ sb)
from outstanding-volatile-writes-unowned-by-others-store-buffer
[OF i-bound ts_{sb}-i out acq]
show ?thesis **by** (simp add: ts_{sb}′ $Read_{sb}$ $\mathcal{O}_{sb}$′)

qed
next
show read-only-reads-unowned ts_{sb}'
proof –
have ro: read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′) $\mathcal{O}_{sb}$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb′)
⊆ read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{O}_{sb}$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
by (auto simp add: sb Read$_{sb}$)
have $\mathcal{O}_{sb}$ ∪ all-acquired sb′ ⊆ $\mathcal{O}_{sb}$ ∪ all-acquired sb
by (auto simp add: sb Read$_{sb}$)
from read-only-reads-unowned-nth-update [OF i-bound ts_{sb}-i ro this]
show ?thesis
by (simp add: ts_{sb}' sb $\mathcal{O}_{sb}'$)
qed
next
show ownership-distinct ts_{sb}'
proof –
have acq: all-acquired sb′ ⊆ all-acquired sb
by (auto simp add: Read$_{sb}$ sb)
with ownership-distinct-instructions-read-value-store-buffer-independent
[OF i-bound ts_{sb}-i]
show ?thesis **by** (simp add: ts_{sb}' Read$_{sb}$ $\mathcal{O}_{sb}'$)
qed
qed

have valid-sharing′: valid-sharing $\mathcal{S}_{sb}'$ ts_{sb}'
proof (intro-locales)
from outstanding-non-volatile-writes-unshared [OF i-bound ts_{sb}-i]
have non-volatile-writes-unshared $\mathcal{S}_{sb}$ sb′
by (auto simp add: sb Read$_{sb}$)
from outstanding-non-volatile-writes-unshared-nth-update [OF i-bound this]
show outstanding-non-volatile-writes-unshared $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: ts_{sb}' $\mathcal{S}_{sb}'$)
next
from sharing-consis [OF i-bound ts_{sb}-i]
have sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ sb′
by (auto simp add: sb Read$_{sb}$)
from sharing-consis-nth-update [OF i-bound this]
show sharing-consis $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: ts_{sb}' $\mathcal{O}_{sb}'$ $\mathcal{S}_{sb}'$)
next
from read-only-unowned-nth-update [OF i-bound read-only-unowned [OF i-bound ts_{sb}-i]
]
show read-only-unowned $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: $\mathcal{S}_{sb}'$ ts_{sb}' $\mathcal{O}_{sb}'$)
next
from unowned-shared-nth-update [OF i-bound ts_{sb}-i subset-refl]
show unowned-shared $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: ts_{sb}' $\mathcal{O}_{sb}'$ $\mathcal{S}_{sb}'$)

next
from no-outstanding-write-to-read-only-memory [OF i-bound ts_{sb}-i]
have no-write-to-read-only-memory $\mathcal{S}_{sb}$ sb′
by (auto simp add: sb $Read_{sb}$)
from no-outstanding-write-to-read-only-memory-nth-update [OF i-bound this]
show no-outstanding-write-to-read-only-memory $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: $\mathcal{S}_{sb}'$ ts_{sb}' sb)
qed

have valid-reads′: valid-reads m_{sb}' ts_{sb}'
proof −
from valid-reads [OF i-bound ts_{sb}-i]
have reads-consistent False $\mathcal{O}_{sb}$ m_{sb} sb′
by (simp add: sb $Read_{sb}$)
from valid-reads-nth-update [OF i-bound this]
show ?thesis **by** (simp add: m_{sb}' ts_{sb}' $\mathcal{O}_{sb}'$)
qed

have valid-program-history′: valid-program-history ts_{sb}'
proof −
from valid-program-history [OF i-bound ts_{sb}-i]
have causal-program-history is_{sb} sb **.**
then have causal′: causal-program-history is_{sb} sb′
by (simp add: sb $Read_{sb}$ causal-program-history-def)

from valid-last-prog [OF i-bound ts_{sb}-i]
have last-prog p_{sb} sb = p_{sb}**.**
hence last-prog p_{sb} sb′ = p_{sb}
by (simp add: sb $Read_{sb}$)

from valid-program-history-nth-update [OF i-bound causal′ this]
show ?thesis
by (simp add: ts_{sb}')
qed

from is-sim
have is-sim: instrs (dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb′) @ is_{sb} =
is @ prog-instrs (dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb′)
by (simp add: sb $Read_{sb}$ suspends)

from valid-history [OF i-bound ts_{sb}-i]
have j_{sb}-v: j_{sb} t = Some v
by (simp add: history-consistent-access-last-read sb $Read_{sb}$ split:option.splits)

have (ts,m,$\mathcal{S}$) $\Rightarrow_d^*$ (ts,m,$\mathcal{S}$) **by** blast

moreover

note flush-commute= flush-all-until-volatile-write-$Read_{sb}$-commute [OF i-bound ts_{sb}-i [simplified sb $Read_{sb}$]]

note share-commute =
share-all-until-volatile-write-update-sb [of sb′ sb, OF - i-bound ts_{sb}-i, simplified sb $Read_{sb}$, simplified]
have (ts_{sb} [i := (p_{sb},is_{sb}, j_{sb}, sb′,$\mathcal{D}_{sb}$, $\mathcal{O}_{sb}$,$\mathcal{R}_{sb}$′)],m_{sb},$\mathcal{S}_{sb}$′) ∼ (ts,m,$\mathcal{S}$)
apply (rule sim-config.intros)
apply (simp add: m flush-commute)
apply (clarsimp simp add: $\mathcal{S}$ $\mathcal{S}_{sb}$′ share-commute)
using leq
apply simp

using i-bound i-bound′ ts-sim ts-i is-sim $\mathcal{D}$
apply (clarsimp simp add: Let-def nth-list-update sb suspends $Read_{sb}$ $\mathcal{S}_{sb}$′ $\mathcal{R}_{sb}$′
split: if-split-asm)
done

ultimately show ?thesis
using valid-own′ valid-hist′ valid-reads′ valid-sharing′ tmps-distinct′ m_{sb}′
valid-dd′ valid-sops′ load-tmps-fresh′ enough-flushs′ valid-sharing′
valid-program-history′ valid′
ts_{sb}′ $\mathcal{O}_{sb}$′ $\mathcal{S}_{sb}$′
by (auto simp del: fun-upd-apply)
next
case ($Prog_{sb}$ p_1 p_2 mis)
from flush this **obtain** m_{sb}′: m_{sb}′=m_{sb} **and**
$\mathcal{O}_{sb}$′: $\mathcal{O}_{sb}$′=$\mathcal{O}_{sb}$ **and** $\mathcal{S}_{sb}$′: $\mathcal{S}_{sb}$′=$\mathcal{S}_{sb}$ **and**
$\mathcal{R}_{sb}$′: $\mathcal{R}_{sb}$′=$\mathcal{R}_{sb}$
by cases (auto simp add: sb)

have valid-own′: valid-ownership $\mathcal{S}_{sb}$′ ts_{sb}′
proof (intro-locales)
show outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}_{sb}$′ ts_{sb}′
proof −
from outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ts_{sb}-i] sb
have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ sb′
by (auto simp add: $Prog_{sb}$)
from outstanding-non-volatile-refs-owned-or-read-only-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}′ $Prog_{sb}$ $\mathcal{O}_{sb}$′ $\mathcal{S}_{sb}$′)
qed
next
show outstanding-volatile-writes-unowned-by-others ts_{sb}′
proof −
from sb
have out: outstanding-refs is-volatile-$Write_{sb}$ sb′ ⊆ outstanding-refs is-volatile-$Write_{sb}$ sb
by (auto simp add: $Prog_{sb}$)
have acq: all-acquired sb′ ⊆ all-acquired sb
by (auto simp add: $Prog_{sb}$ sb)
from outstanding-volatile-writes-unowned-by-others-store-buffer
[OF i-bound ts_{sb}-i out acq]

show ?thesis **by** (simp add: ts_{sb}' $Prog_{sb}$ $\mathcal{O}_{sb}'$)
qed
next
show read-only-reads-unowned ts_{sb}'
proof –
have ro: read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb′) $\mathcal{O}_{sb}$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb′)
⊆ read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{O}_{sb}$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
by (auto simp add: sb $Prog_{sb}$)
have $\mathcal{O}_{sb}$ ∪ all-acquired sb′ ⊆ $\mathcal{O}_{sb}$ ∪ all-acquired sb
by (auto simp add: sb $Prog_{sb}$)
from read-only-reads-unowned-nth-update [OF i-bound ts_{sb}-i ro this]
show ?thesis
by (simp add: ts_{sb}' sb $\mathcal{O}_{sb}'$)
qed
next
show ownership-distinct ts_{sb}'
proof –
have acq: all-acquired sb′ ⊆ all-acquired sb
by (auto simp add: $Prog_{sb}$ sb)
with ownership-distinct-instructions-read-value-store-buffer-independent
[OF i-bound ts_{sb}-i]
show ?thesis **by** (simp add: ts_{sb}' $Prog_{sb}$ $\mathcal{O}_{sb}'$)
qed
qed

have valid-sharing′: valid-sharing $\mathcal{S}_{sb}'$ ts_{sb}'
proof (intro-locales)
from outstanding-non-volatile-writes-unshared [OF i-bound ts_{sb}-i]
have non-volatile-writes-unshared $\mathcal{S}_{sb}$ sb′
by (auto simp add: sb $Prog_{sb}$)
from outstanding-non-volatile-writes-unshared-nth-update [OF i-bound this]
show outstanding-non-volatile-writes-unshared $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: ts_{sb}' $\mathcal{S}_{sb}'$)
next
from sharing-consis [OF i-bound ts_{sb}-i]
have sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ sb′
by (auto simp add: sb $Prog_{sb}$)
from sharing-consis-nth-update [OF i-bound this]
show sharing-consis $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: ts_{sb}' $\mathcal{O}_{sb}'$ $\mathcal{S}_{sb}'$)
next
from read-only-unowned-nth-update [OF i-bound read-only-unowned [OF i-bound ts_{sb}-i]
]
show read-only-unowned $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: $\mathcal{S}_{sb}'$ ts_{sb}' $\mathcal{O}_{sb}'$)
next
from unowned-shared-nth-update [OF i-bound ts_{sb}-i subset-refl]
show unowned-shared $\mathcal{S}_{sb}'$ ts_{sb}'

by (simp add: ts_{sb}' $\mathcal{O}_{sb}'$ $\mathcal{S}_{sb}'$)
next
from no-outstanding-write-to-read-only-memory [OF i-bound ts_{sb}-i]
have no-write-to-read-only-memory $\mathcal{S}_{sb}$ sb′
by (auto simp add: sb $Prog_{sb}$)
from no-outstanding-write-to-read-only-memory-nth-update [OF i-bound this]
show no-outstanding-write-to-read-only-memory $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: $\mathcal{S}_{sb}'$ ts_{sb}' sb)
qed

have valid-reads′: valid-reads m_{sb}' ts_{sb}'
proof −
from valid-reads [OF i-bound ts_{sb}-i]
have reads-consistent False $\mathcal{O}_{sb}$ m_{sb} sb′
by (simp add: sb $Prog_{sb}$)
from valid-reads-nth-update [OF i-bound this]
show ?thesis **by** (simp add: m_{sb}' ts_{sb}' $\mathcal{O}_{sb}'$)
qed

have valid-program-history′: valid-program-history ts_{sb}'
proof −
from valid-program-history [OF i-bound ts_{sb}-i]
have causal-program-history is_{sb} sb **.**
then have causal′: causal-program-history is_{sb} sb′
by (simp add: sb $Prog_{sb}$ causal-program-history-def)

from valid-last-prog [OF i-bound ts_{sb}-i]
have last-prog p_{sb} sb = p_{sb}**.**
hence last-prog p_2 sb′ = p_{sb}
by (simp add: sb $Prog_{sb}$)
from last-prog-to-last-prog-same [OF this]
have last-prog p_{sb} sb′ = p_{sb}**.**

from valid-program-history-nth-update [OF i-bound causal′ this]
show ?thesis
by (simp add: ts_{sb}')
qed

from is-sim
have is-sim: instrs (dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb′) @ is_{sb} =
is @ prog-instrs (dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb′)
by (simp add: suspends sb $Prog_{sb}$)

have (ts,m,$\mathcal{S}$) $\Rightarrow_d^*$ (ts,m,$\mathcal{S}$) **by** blast

moreover

note flush-commute = flush-all-until-volatile-write-$Prog_{sb}$-commute [OF i-bound ts_{sb}-i [simplified sb $Prog_{sb}$]]

note share-commute =
share-all-until-volatile-write-update-sb [of sb′ sb, OF - i-bound ts_{sb}-i, simplified sb $Prog_{sb}$, simplified]

have (ts_{sb} [i := (p_{sb},is_{sb}, j_{sb}, sb′,$\mathcal{D}_{sb}$, $\mathcal{O}_{sb}$,$\mathcal{R}_{sb}$)],m_{sb},$\mathcal{S}_{sb}$′) ∼ (ts,m,$\mathcal{S}$)
apply (rule sim-config.intros)
apply (simp add: m flush-commute)
apply (clarsimp simp add: $\mathcal{S}$ $\mathcal{S}_{sb}$′ share-commute)
using leq
apply simp

using i-bound i-bound′ ts-sim ts-i is-sim $\mathcal{D}$
apply (clarsimp simp add: Let-def nth-list-update sb suspends $Prog_{sb}$ $\mathcal{R}_{sb}$′ $\mathcal{S}_{sb}$′
split: if-split-asm)
done
ultimately show ?thesis
using valid-own′ valid-hist′ valid-reads′ valid-sharing′ tmps-distinct′ m_{sb}′
valid-dd′ valid-sops′ load-tmps-fresh′ enough-flushs′ valid-sharing′
valid-program-history′ valid′
ts_{sb}′ $\mathcal{S}_{sb}$′ $\mathcal{O}_{sb}$′ $\mathcal{R}_{sb}$′ $\mathcal{S}_{sb}$′
by (auto simp del: fun-upd-apply)
next
case ($Ghost_{sb}$ A L R W)
from flush $Ghost_{sb}$
obtain
$\mathcal{O}_{sb}$′: $\mathcal{O}_{sb}$′=$\mathcal{O}_{sb}$ ∪ A − R **and**
$\mathcal{S}_{sb}$′: $\mathcal{S}_{sb}$′=$\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L **and**
$\mathcal{R}_{sb}$′: $\mathcal{R}_{sb}$′= augment-rels (dom $\mathcal{S}_{sb}$) R $\mathcal{R}_{sb}$ **and**
m_{sb}′: m_{sb}′=m_{sb}
by cases (auto simp add: sb)

from sharing-consis [OF i-bound ts_{sb}-i]
obtain
A-shared-owned: A ⊆ dom $\mathcal{S}_{sb}$ ∪ $\mathcal{O}_{sb}$ **and**
L-subset: L ⊆ A **and**
A-R: A ∩ R = {} **and**
R-owned: R ⊆ $\mathcal{O}_{sb}$
by (clarsimp simp add: sb $Ghost_{sb}$)

have valid-own′: valid-ownership $\mathcal{S}_{sb}$′ ts_{sb}′
proof (intro-locales)
show outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}_{sb}$′ ts_{sb}′
proof
fix j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ acq_j j_j sb_j p_j
assume j-bound: j < length ts_{sb}′
assume ts_{sb}′-j: ts_{sb}′!j = (p_j,is_j,j_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$′ $\mathcal{O}_j$ sb_j
proof (cases j=i)
case True
from outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ts_{sb}-i]

have non-volatile-owned-or-read-only False ($\mathcal{S}_{\mathsf{sb}}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ($\mathcal{O}_{\mathsf{sb}}$ $\cup$ A $-$ R) sb′
by (auto simp add: sb Ghost$_{\mathsf{sb}}$ non-volatile-owned-or-read-only-pending-write-antimono)
then show ?thesis
using True i-bound ts$_{\mathsf{sb}}$′-j
by (auto simp add: ts$_{\mathsf{sb}}$′ $\mathcal{S}_{\mathsf{sb}}$′ sb $\mathcal{O}_{\mathsf{sb}}$′)
next
case False
from j-bound **have** j-bound′: j < length ts$_{\mathsf{sb}}$
by (auto simp add: ts$_{\mathsf{sb}}$′)
with ts$_{\mathsf{sb}}$′-j False i-bound
have ts$_{\mathsf{sb}}$-j: ts$_{\mathsf{sb}}$!j = (p$_j$,is$_j$,j$_j$,sb$_j$,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (auto simp add: ts$_{\mathsf{sb}}$′)

note nvo = outstanding-non-volatile-refs-owned-or-read-only [OF j-bound′ ts$_{\mathsf{sb}}$-j]

from read-only-unowned [OF i-bound ts$_{\mathsf{sb}}$-i] R-owned
have R $\cap$ read-only $\mathcal{S}_{\mathsf{sb}}$ = {}
by auto

with read-only-reads-unowned [OF j-bound′ i-bound False ts$_{\mathsf{sb}}$-j ts$_{\mathsf{sb}}$-i] L-subset
have $\forall$ a$\in$read-only-reads
(acquired True (takeWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_j$) $\mathcal{O}_j$)
(dropWhile (Not $\circ$ is-volatile-Write$_{\mathsf{sb}}$) sb$_j$).
a $\in$ read-only $\mathcal{S}_{\mathsf{sb}}$ $\longrightarrow$ a $\in$ read-only ($\mathcal{S}_{\mathsf{sb}}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (auto simp add: in-read-only-convs sb Ghost$_{\mathsf{sb}}$)
from non-volatile-owned-or-read-only-read-only-reads-eq′ [OF nvo this]
have non-volatile-owned-or-read-only False ($\mathcal{S}_{\mathsf{sb}}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $\mathcal{O}_j$ sb$_j$.
thus ?thesis **by** (simp add: $\mathcal{S}_{\mathsf{sb}}$′)
qed
qed
next
show outstanding-volatile-writes-unowned-by-others ts$_{\mathsf{sb}}$′
proof (unfold-locales)
fix i$_1$ j p$_1$ is$_1$ $\mathcal{O}_1$ $\mathcal{R}_1$ $\mathcal{D}_1$ xs$_1$ sb$_1$ p$_j$ is$_j$ $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs$_j$ sb$_j$
assume i$_1$-bound: i$_1$ < length ts$_{\mathsf{sb}}$′
assume j-bound: j < length ts$_{\mathsf{sb}}$′
assume i$_1$-j: i$_1$ $\neq$ j
assume ts-i$_1$: ts$_{\mathsf{sb}}$′!i$_1$ = (p$_1$,is$_1$,xs$_1$,sb$_1$,$\mathcal{D}_1$,$\mathcal{O}_1$,$\mathcal{R}_1$)
assume ts-j: ts$_{\mathsf{sb}}$′!j = (p$_j$,is$_j$,xs$_j$,sb$_j$,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show ($\mathcal{O}_j$ $\cup$ all-acquired sb$_j$) $\cap$ outstanding-refs is-volatile-Write$_{\mathsf{sb}}$ sb$_1$ = {}
proof (cases i$_1$=i)
case True
from i$_1$-j True **have** neq-i-j: i$\neq$j
by auto
from j-bound **have** j-bound′: j < length ts$_{\mathsf{sb}}$
by (simp add: ts$_{\mathsf{sb}}$′)
from ts-j neq-i-j **have** ts-j′: ts$_{\mathsf{sb}}$!j = (p$_j$,is$_j$,xs$_j$,sb$_j$,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (simp add: ts$_{\mathsf{sb}}$′)
from outstanding-volatile-writes-unowned-by-others [OF i-bound j-bound′ neq-i-j

ts_{sb}-i ts-j′] ts-i_1 i-bound ts_{sb}-i True **show** ?thesis
by (clarsimp simp add: ts_{sb}′ sb $Ghost_{sb}$)
next
case False
note i_1-i = this
from i_1-bound **have** i_1-bound′: i_1 < length ts_{sb}
by (simp add: ts_{sb}′ sb)
hence i_1-bound″: i_1 < length (map owned ts_{sb})
by auto
from ts-i_1 False **have** ts-i_1′: ts_{sb}!i_1 = (p_1,is_1,xs_1,sb_1,$\mathcal{D}_1$,$\mathcal{O}_1$,$\mathcal{R}_1$)
by (simp add: ts_{sb}′ sb)
show ?thesis
proof (cases j=i)
case True
from outstanding-volatile-writes-unowned-by-others [OF i_1-bound′ i-bound i_1-i ts-i_1′
ts_{sb}-i]
have ($\mathcal{O}_{sb}$ ∪ all-acquired sb) ∩ outstanding-refs is-volatile-$Write_{sb}$ sb_1 = {}.
then show ?thesis
using True i_1-i ts-j ts_{sb}-i i-bound
by (auto simp add: sb $Ghost_{sb}$ ts_{sb}′ $\mathcal{O}_{sb}$′)
next
case False
from j-bound **have** j-bound′: j < length ts_{sb}
by (simp add: ts_{sb}′)
from ts-j False **have** ts-j′: ts_{sb}!j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (simp add: ts_{sb}′)
from outstanding-volatile-writes-unowned-by-others
[OF i_1-bound′ j-bound′ i_1-j ts-i_1′ ts-j′]
show ($\mathcal{O}_j$ ∪ all-acquired sb_j) ∩ outstanding-refs is-volatile-$Write_{sb}$ sb_1 = {} .
qed
qed
qed
next
show read-only-reads-unowned ts_{sb}′
proof
fix n m
fix p_n is_n $\mathcal{O}_n$ $\mathcal{R}_n$ $\mathcal{D}_n$ j_n sb_n p_m is_m $\mathcal{O}_m$ $\mathcal{R}_m$ $\mathcal{D}_m$ j_m sb_m
assume n-bound: n < length ts_{sb}′
and m-bound: m < length ts_{sb}′
and neq-n-m: n≠m
and nth: ts_{sb}′!n = (p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
and mth: ts_{sb}′!m =(p_m, is_m, j_m, sb_m, $\mathcal{D}_m$, $\mathcal{O}_m$,$\mathcal{R}_m$)
from n-bound **have** n-bound′: n < length ts_{sb} **by** (simp add: ts_{sb}′)
from m-bound **have** m-bound′: m < length ts_{sb} **by** (simp add: ts_{sb}′)
show ($\mathcal{O}_m$ ∪ all-acquired sb_m) ∩
read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n) $\mathcal{O}_n$)
(dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n) =
{}
proof (cases m=i)
case True

with neq-n-m **have** neq-n-i: n≠i
by auto

with n-bound nth i-bound **have** nth′: ts_{sb}!n =(p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
by (auto simp add: ts_{sb}′)
note read-only-reads-unowned [OF n-bound′ i-bound neq-n-i nth′ ts_{sb}-i]
then
show ?thesis
using True ts_{sb}-i neq-n-i nth mth n-bound′ m-bound′ L-subset
by (auto simp add: ts_{sb}′ $\mathcal{O}_{sb}$′ sb $Ghost_{sb}$)
next
case False
note neq-m-i = this
with m-bound mth i-bound **have** mth′: ts_{sb}!m = (p_m, is_m, j_m, sb_m, $\mathcal{D}_m$, $\mathcal{O}_m$,$\mathcal{R}_m$)
by (auto simp add: ts_{sb}′)
show ?thesis
proof (cases n=i)
case True
from read-only-reads-append [of ($\mathcal{O}_{sb}$ ∪ A − R) (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n)
(dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n)]
have read-only-reads
(acquired True (takeWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n) ($\mathcal{O}_{sb}$ ∪ A − R))
(dropWhile (Not ∘ is-volatile-$Write_{sb}$) sb_n) ⊆ read-only-reads ($\mathcal{O}_{sb}$ ∪ A − R) sb_n
by auto

with ts_{sb}-i nth mth neq-m-i n-bound′ True
read-only-reads-unowned [OF i-bound m-bound′ False [symmetric] ts_{sb}-i mth′]
show ?thesis
by (auto simp add: ts_{sb}′ sb $\mathcal{O}_{sb}$′ $Ghost_{sb}$)
next
case False
with n-bound nth i-bound **have** nth′: ts_{sb}!n =(p_n, is_n, j_n, sb_n, $\mathcal{D}_n$, $\mathcal{O}_n$,$\mathcal{R}_n$)
by (auto simp add: ts_{sb}′)
from read-only-reads-unowned [OF n-bound′ m-bound′ neq-n-m nth′ mth′] False neq-m-i
show ?thesis
by (clarsimp)
qed
qed
qed
next
show ownership-distinct ts_{sb}′
proof (unfold-locales)
fix i_1 j p_1 is_1 $\mathcal{O}_1$ $\mathcal{R}_1$ $\mathcal{D}_1$ xs_1 sb_1 p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ xs_j sb_j
assume i_1-bound: i_1 < length ts_{sb}′
assume j-bound: j < length ts_{sb}′
assume i_1-j: i_1 ≠ j
assume ts-i_1: ts_{sb}′!i_1 = (p_1,is_1,xs_1,sb_1,$\mathcal{D}_1$,$\mathcal{O}_1$,$\mathcal{R}_1$)

assume ts-j: ts_{sb}′!j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show ($\mathcal{O}_1$ ∪ all-acquired sb_1) ∩ ($\mathcal{O}_j$ ∪ all-acquired sb_j)= {}
proof (cases i_1=i)
case True
with i_1-j **have** i-j: i≠j
by simp

from j-bound **have** j-bound′: j < length ts_{sb}
by (simp add: ts_{sb}′)
hence j-bound″: j < length (map owned ts_{sb})
by simp
from ts-j i-j **have** ts-j′: ts_{sb}!j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (simp add: ts_{sb}′)

from ownership-distinct [OF i-bound j-bound′ i-j ts_{sb}-i ts-j′]
show ?thesis
using ts_{sb}-i True ts-i_1 i-bound $\mathcal{O}_{sb}$′
by (auto simp add: ts_{sb}′ sb $Ghost_{sb}$)
next
case False
note i_1-i = this
from i_1-bound **have** i_1-bound′: i_1 < length ts_{sb}
by (simp add: ts_{sb}′)
hence i_1-bound″: i_1 < length (map owned ts_{sb})
by simp
from ts-i_1 False **have** ts-i_1′: ts_{sb}!i_1 = (p_1,is_1,xs_1,sb_1,$\mathcal{D}_1$,$\mathcal{O}_1$,$\mathcal{R}_1$)
by (simp add: ts_{sb}′)
show ?thesis
proof (cases j=i)
case True
from ownership-distinct [OF i_1-bound′ i-bound i_1-i ts-i_1′ ts_{sb}-i]
show ?thesis
using ts_{sb}-i True ts-j i-bound $\mathcal{O}_{sb}$′
by (auto simp add: ts_{sb}′ sb $Ghost_{sb}$)
next
case False
from j-bound **have** j-bound′: j < length ts_{sb}
by (simp add: ts_{sb}′)
from ts-j False **have** ts-j′: ts_{sb}!j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (simp add: ts_{sb}′)
from ownership-distinct [OF i_1-bound′ j-bound′ i_1-j ts-i_1′ ts-j′]
show ?thesis **.**
qed
qed
qed
qed

have valid-sharing′: valid-sharing ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) ts_{sb}′
proof (intro-locales)
show outstanding-non-volatile-writes-unshared ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) ts_{sb}′

proof (unfold-locales)
fix j p_j is_j $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ acq_j xs_j sb_j
assume j-bound: j < length ts_{sb}'
assume jth: ts_{sb}' ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show non-volatile-writes-unshared ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) sb_j
proof (cases i=j)
case True
with outstanding-non-volatile-writes-unshared [OF i-bound ts_{sb}-i]
i-bound jth ts_{sb}-i **show** ?thesis
by (clarsimp simp add: ts_{sb}' sb $Ghost_{sb}$)
next
case False
from j-bound **have** j-bound′: j < length ts_{sb}
by (auto simp add: ts_{sb}')
from jth False **have** jth′: ts_{sb} ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (auto simp add: ts_{sb}')
from j-bound jth i-bound False
have j: non-volatile-writes-unshared $\mathcal{S}_{sb}$ sb_j
apply –
apply (rule outstanding-non-volatile-writes-unshared)
apply (auto simp add: ts_{sb}')
done
from jth False **have** jth′: ts_{sb} ! j = (p_j,is_j,xs_j,sb_j,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (auto simp add: ts_{sb}')
from outstanding-non-volatile-writes-unshared [OF j-bound′ jth′]
have unshared: non-volatile-writes-unshared $\mathcal{S}_{sb}$ sb_j.

have ∀ a∈dom ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) – dom $\mathcal{S}_{sb}$. a ∉ outstanding-refs is-non-volatile-$Write_{sb}$ sb_j
proof –
{
fix a
assume a-in: a ∈ dom ($\mathcal{S}_{sb}$ $\oplus_W$ R $\ominus_A$ L) – dom $\mathcal{S}_{sb}$
hence a-R: a ∈ R
by clarsimp
assume a-in-j: a ∈ outstanding-refs is-non-volatile-$Write_{sb}$ sb_j
have False
proof –
from non-volatile-owned-or-read-only-outstanding-non-volatile-writes [OF
outstanding-non-volatile-refs-owned-or-read-only [OF j-bound′ jth′]]
a-in-j
have a ∈ $\mathcal{O}_j$ ∪ all-acquired sb_j
by auto

moreover
with ownership-distinct [OF i-bound j-bound′ False ts_{sb}-i jth′] a-R R-owned
show False
by blast
qed
}

thus ?thesis **by** blast
qed

from non-volatile-writes-unshared-no-outstanding-non-volatile-Write$_{\mathsf{sb}}$
[OF unshared this]
show ?thesis **.**
qed
qed
next
show sharing-consis ($\mathcal{S}_{\mathsf{sb}} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ts$_{\mathsf{sb}}'$
proof (unfold-locales)
fix j p$_j$ is$_j$ $\mathcal{O}_j$ $\mathcal{R}_j$ $\mathcal{D}_j$ acq$_j$ xs$_j$ sb$_j$
assume j-bound: j < length ts$_{\mathsf{sb}}'$
assume jth: ts$_{\mathsf{sb}}'$! j = (p$_j$,is$_j$,xs$_j$,sb$_j$,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
show sharing-consistent ($\mathcal{S}_{\mathsf{sb}} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $\mathcal{O}_j$ sb$_j$
proof (cases i=j)
case True
with i-bound jth ts$_{\mathsf{sb}}$-i sharing-consis [OF i-bound ts$_{\mathsf{sb}}$-i]
show ?thesis
by (clarsimp simp add: ts$_{\mathsf{sb}}'$ sb Ghost$_{\mathsf{sb}}$ $\mathcal{O}_{\mathsf{sb}}'$)
next
case False
from j-bound **have** j-bound$'$: j < length ts$_{\mathsf{sb}}$
by (auto simp add: ts$_{\mathsf{sb}}'$)
from jth False **have** jth$'$: ts$_{\mathsf{sb}}$! j = (p$_j$,is$_j$,xs$_j$,sb$_j$,$\mathcal{D}_j$,$\mathcal{O}_j$,$\mathcal{R}_j$)
by (auto simp add: ts$_{\mathsf{sb}}'$)
from sharing-consis [OF j-bound$'$ jth$'$]
have consis: sharing-consistent $\mathcal{S}_{\mathsf{sb}}$ $\mathcal{O}_j$ sb$_j$**.**

have acq-cond: all-acquired sb$_j$ $\cap$ dom $\mathcal{S}_{\mathsf{sb}}$ $-$ dom ($\mathcal{S}_{\mathsf{sb}} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = {}
proof $-$
{
fix a
assume a-acq: a $\in$ all-acquired sb$_j$
assume a $\in$ dom $\mathcal{S}_{\mathsf{sb}}$
assume a-L: a $\in$ L
have False
proof $-$
from ownership-distinct [OF i-bound j-bound$'$ False ts$_{\mathsf{sb}}$-i jth$'$]
have A $\cap$ all-acquired sb$_j$ = {}
by (auto simp add: sb Ghost$_{\mathsf{sb}}$)
with a-acq a-L L-subset
show False
by blast
qed
}
thus ?thesis
by auto

```
    qed

    have uns-cond: all-unshared sbj ∩ dom (𝒮sb ⊕W R ⊖A L) − dom 𝒮sb = {}
    proof −
      {
 fix a
 assume a-uns: a ∈ all-unshared sbj
 assume a ∉ L
 assume a-R:  a ∈ R
 have False
        proof −
   from unshared-acquired-or-owned [OF consis] a-uns
   have a ∈ all-acquired sbj ∪ 𝒪j by auto
   with ownership-distinct [OF i-bound j-bound′ False tssb-i jth′]  R-owned a-R
   show False
     by blast
   qed
      }
      thus ?thesis
        by auto
    qed

    from sharing-consistent-preservation [OF consis acq-cond uns-cond]
    show ?thesis
      by (simp add: tssb′)
  qed
qed
     next
show unowned-shared (𝒮sb ⊕W R ⊖A L) tssb′
proof (unfold-locales)
  show − ⋃((λ(-,-, -, -,-, 𝒪,-). 𝒪) ‘ set tssb′) ⊆ dom (𝒮sb ⊕W R ⊖A L)
  proof −

    have s: ⋃((λ(-,-, -, -,-, 𝒪,-). 𝒪) ‘ set tssb′) =
             ⋃((λ(-,-, -, -,-, 𝒪,-). 𝒪) ‘ set tssb) ∪ A − R

      apply (unfold tssb′ 𝒪sb′)
      apply (rule acquire-release-ownership-nth-update [OF R-owned i-bound tssb-i])
      apply (rule local.ownership-distinct-axioms)
      done

    note unowned-shared L-subset A-R
    then
    show ?thesis
      apply (simp only: s)
      apply auto
      done
  qed
qed
     next
```

show read-only-unowned ($\mathcal{S}_{\mathsf{sb}} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $\mathrm{ts}_{\mathsf{sb}}'$
proof
  fix j $\mathrm{p_j}$ $\mathrm{is_j}$ $\mathcal{O}_{\mathsf{j}}$ $\mathcal{R}_{\mathsf{j}}$ $\mathcal{D}_{\mathsf{j}}$ $\mathrm{xs_j}$ $\mathrm{sb_j}$
  assume j-bound: j < length $\mathrm{ts}_{\mathsf{sb}}'$
  assume jth: $\mathrm{ts}_{\mathsf{sb}}'$! j = ($\mathrm{p_j}$,$\mathrm{is_j}$,$\mathrm{xs_j}$,$\mathrm{sb_j}$,$\mathcal{D}_{\mathsf{j}}$,$\mathcal{O}_{\mathsf{j}}$,$\mathcal{R}_{\mathsf{j}}$)
  show $\mathcal{O}_{\mathsf{j}}$ ∩ read-only ($\mathcal{S}_{\mathsf{sb}} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = {}
  proof (cases i=j)
    case True
    from read-only-unowned [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i]
    have ($\mathcal{O}_{\mathsf{sb}}$ ∪ A − R) ∩ read-only ($\mathcal{S}_{\mathsf{sb}} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) = {}
      by (auto simp add: in-read-only-convs)
    with jth $\mathrm{ts}_{\mathsf{sb}}$-i i-bound True
    show ?thesis
      by (auto simp add: $\mathcal{O}_{\mathsf{sb}}'$ $\mathrm{ts}_{\mathsf{sb}}'$)
  next
    case False
    from j-bound **have** j-bound′: j < length $\mathrm{ts}_{\mathsf{sb}}$
      by (auto simp add: $\mathrm{ts}_{\mathsf{sb}}'$)
    with False jth **have** jth′: $\mathrm{ts}_{\mathsf{sb}}$! j = ($\mathrm{p_j}$,$\mathrm{is_j}$,$\mathrm{xs_j}$,$\mathrm{sb_j}$,$\mathcal{D}_{\mathsf{j}}$,$\mathcal{O}_{\mathsf{j}}$,$\mathcal{R}_{\mathsf{j}}$)
      by (auto simp add: $\mathrm{ts}_{\mathsf{sb}}'$)
    from read-only-unowned [OF j-bound′ jth′]
    have $\mathcal{O}_{\mathsf{j}}$ ∩ read-only $\mathcal{S}_{\mathsf{sb}}$ = {}.
    moreover
    from ownership-distinct [OF i-bound j-bound′ False $\mathrm{ts}_{\mathsf{sb}}$-i jth′] R-owned
    have ($\mathcal{O}_{\mathsf{sb}}$ ∪ A) ∩ $\mathcal{O}_{\mathsf{j}}$ = {}
      by (auto simp add: sb $\mathrm{Ghost}_{\mathsf{sb}}$)
    moreover note R-owned A-R
    ultimately show ?thesis
      by (fastforce simp add: in-read-only-convs split: if-split-asm)
  qed
qed
     next
show no-outstanding-write-to-read-only-memory ($\mathcal{S}_{\mathsf{sb}} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $\mathrm{ts}_{\mathsf{sb}}'$
proof
  fix j $\mathrm{p_j}$ $\mathrm{is_j}$ $\mathcal{O}_{\mathsf{j}}$ $\mathcal{R}_{\mathsf{j}}$ $\mathcal{D}_{\mathsf{j}}$ $\mathrm{xs_j}$ $\mathrm{sb_j}$
  assume j-bound: j < length $\mathrm{ts}_{\mathsf{sb}}'$
  assume jth: $\mathrm{ts}_{\mathsf{sb}}'$! j = ($\mathrm{p_j}$,$\mathrm{is_j}$,$\mathrm{xs_j}$,$\mathrm{sb_j}$,$\mathcal{D}_{\mathsf{j}}$,$\mathcal{O}_{\mathsf{j}}$,$\mathcal{R}_{\mathsf{j}}$)
  show no-write-to-read-only-memory ($\mathcal{S}_{\mathsf{sb}} \oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) $\mathrm{sb_j}$
  proof (cases i=j)
    case True
    with jth $\mathrm{ts}_{\mathsf{sb}}$-i i-bound no-outstanding-write-to-read-only-memory [OF i-bound $\mathrm{ts}_{\mathsf{sb}}$-i]
    show ?thesis
      by (auto simp add: sb $\mathrm{ts}_{\mathsf{sb}}'$ $\mathrm{Ghost}_{\mathsf{sb}}$)
  next
    case False
    from j-bound **have** j-bound′: j < length $\mathrm{ts}_{\mathsf{sb}}$
      by (auto simp add: $\mathrm{ts}_{\mathsf{sb}}'$)
    with False jth **have** jth′: $\mathrm{ts}_{\mathsf{sb}}$! j = ($\mathrm{p_j}$,$\mathrm{is_j}$,$\mathrm{xs_j}$,$\mathrm{sb_j}$,$\mathcal{D}_{\mathsf{j}}$,$\mathcal{O}_{\mathsf{j}}$,$\mathcal{R}_{\mathsf{j}}$)
      by (auto simp add: $\mathrm{ts}_{\mathsf{sb}}'$)
    from no-outstanding-write-to-read-only-memory [OF j-bound′ jth′]

have nw: no-write-to-read-only-memory $\mathcal{S}_{\mathsf{sb}}$ $\mathrm{sb_j}$.

have R $\cap$ outstanding-refs is-Write$_{\mathsf{sb}}$ $\mathrm{sb_j}$ = {}
proof −
note dist = ownership-distinct [OF i-bound j-bound′ False ts$_{\mathsf{sb}}$-i jth′]
from non-volatile-owned-or-read-only-outstanding-non-volatile-writes
[OF outstanding-non-volatile-refs-owned-or-read-only [OF j-bound′ jth′]]
dist
have outstanding-refs is-non-volatile-Write$_{\mathsf{sb}}$ $\mathrm{sb_j}$ $\cap$ $\mathcal{O}_{\mathsf{sb}}$ = {}
by auto
moreover
from outstanding-volatile-writes-unowned-by-others [OF j-bound′ i-bound
False [symmetric] jth′ ts$_{\mathsf{sb}}$-i]
have outstanding-refs is-volatile-Write$_{\mathsf{sb}}$ $\mathrm{sb_j}$ $\cap$ $\mathcal{O}_{\mathsf{sb}}$ = {}
by auto
ultimately have outstanding-refs is-Write$_{\mathsf{sb}}$ $\mathrm{sb_j}$ $\cap$ $\mathcal{O}_{\mathsf{sb}}$ = {}
by (auto simp add: misc-outstanding-refs-convs)
with R-owned
show ?thesis **by** blast
qed
then
have ∀ a∈outstanding-refs is-Write$_{\mathsf{sb}}$ $\mathrm{sb_j}$.
a ∈ read-only ($\mathcal{S}_{\mathsf{sb}}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L) ⟶ a ∈ read-only $\mathcal{S}_{\mathsf{sb}}$
by (auto simp add: in-read-only-convs)

from no-write-to-read-only-memory-read-only-reads-eq [OF nw this]
show ?thesis .
qed
qed
qed

have valid-reads′: valid-reads $\mathrm{m_{sb}}'$ $\mathrm{ts_{sb}}'$
proof −
from valid-reads [OF i-bound ts$_{\mathsf{sb}}$-i]
have reads-consistent False ($\mathcal{O}_{\mathsf{sb}}$ ∪ A − R) $\mathrm{m_{sb}}$ sb′
by (simp add: sb Ghost$_{\mathsf{sb}}$)
from valid-reads-nth-update [OF i-bound this]
show ?thesis **by** (simp add: $\mathrm{m_{sb}}'$ $\mathrm{ts_{sb}}'$ $\mathcal{O}_{\mathsf{sb}}'$)
qed

have valid-program-history′: valid-program-history $\mathrm{ts_{sb}}'$
proof −
from valid-program-history [OF i-bound ts$_{\mathsf{sb}}$-i]
have causal-program-history $\mathrm{is_{sb}}$ sb .
then have causal′: causal-program-history $\mathrm{is_{sb}}$ sb′
by (simp add: sb Ghost$_{\mathsf{sb}}$ causal-program-history-def)

from valid-last-prog [OF i-bound ts$_{\mathsf{sb}}$-i]
have last-prog $\mathrm{p_{sb}}$ sb = $\mathrm{p_{sb}}$.
hence last-prog $\mathrm{p_{sb}}$ sb′ = $\mathrm{p_{sb}}$

by (simp add: sb $\text{Ghost}_{\mathsf{sb}}$)

from valid-program-history-nth-update [OF i-bound causal′ this]
show ?thesis
by (simp add: $\text{ts}_{\mathsf{sb}}{}'$)
qed

from is-sim
have is-sim: instrs (dropWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb′) @ is_{sb} =
is @ prog-instrs (dropWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb′)
by (simp add: sb $\text{Ghost}_{\mathsf{sb}}$ suspends)

have (ts,m,$\mathcal{S}$) $\Rightarrow_{\mathsf{d}}{}^*$ (ts,m,$\mathcal{S}$) **by** blast
moreover

note flush-commute =
flush-all-until-volatile-write-$\text{Ghost}_{\mathsf{sb}}$-commute [OF i-bound ts_{sb}-i [simplified sb $\text{Ghost}_{\mathsf{sb}}$]]

have dist-R-L-A: ∀ j p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb.
j < length ts_{sb} ⟶ i≠ j⟶
ts_{sb} ! j = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$) ⟶
(all-shared (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) ∪
all-unshared (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb) ∪
all-acquired (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb)) ∩ (R ∪ L ∪ A) = {}
proof −
{
fix j p_{j} is_{j} $\mathcal{O}_{\text{j}}$ $\mathcal{R}_{\text{j}}$ $\mathcal{D}_{\text{j}}$ j_{j} sb_{j} x
assume j-bound: j < length ts_{sb}
assume neq-i-j: i ≠ j
assume jth: ts_{sb}!j = (p_{j},is_{j}, j_{j},sb_{j},$\mathcal{D}_{\text{j}}$,$\mathcal{O}_{\text{j}}$,$\mathcal{R}_{\text{j}}$)
assume x-shared: x ∈ all-shared (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{j}) ∪
all-unshared (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{j}) ∪
all-acquired (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{j})
assume x-R-L-A: x ∈ R ∪ L ∪ A
have False
proof −
from x-shared all-shared-acquired-or-owned [OF sharing-consis [OF j-bound jth]]
unshared-acquired-or-owned [OF sharing-consis [OF j-bound jth]]
all-shared-append [of (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{j}) (dropWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{j})]
all-unshared-append [of (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{j}) (dropWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{j})]
all-acquired-append [of (takeWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{j}) (dropWhile (Not ∘ is-volatile-$\text{Write}_{\mathsf{sb}}$) sb_{j})]
have x ∈ all-acquired sb_{j} ∪ $\mathcal{O}_{\text{j}}$
by auto
moreover
from x-R-L-A R-owned L-subset
have x ∈ all-acquired sb ∪ $\mathcal{O}_{\mathsf{sb}}$

```
          by (auto simp add: sb Ghost_sb)
        moreover
        note ownership-distinct [OF i-bound j-bound neq-i-j ts_sb-i jth]
        ultimately show False by blast
      qed
    }
    thus ?thesis by blast
  qed

  {
fix j p_j is_j 𝒪_j ℛ_j 𝒟_j j_j sb_j x
assume jth: ts_sb!j = (p_j,is_j,j_j,sb_j,𝒟_j,𝒪_j,ℛ_j)
assume j-bound: j < length ts_sb
    assume neq: i ≠ j
    have release (takeWhile (Not ∘ is-volatile-Write_sb) sb_j)
                    (dom 𝒮_sb ∪ R − L) ℛ_j
        = release (takeWhile (Not ∘ is-volatile-Write_sb) sb_j)
                    (dom 𝒮_sb) ℛ_j
    proof −
      {
        fix a
        assume a-in: a ∈ all-shared (takeWhile (Not ∘ is-volatile-Write_sb) sb_j)
        have (a ∈ (dom 𝒮_sb ∪ R − L)) = (a ∈ dom 𝒮_sb)
        proof −
          from ownership-distinct [OF i-bound j-bound neq  ts_sb-i jth]

          have A-dist: A ∩ (𝒪_j ∪ all-acquired sb_j) = {}
            by (auto simp add: sb Ghost_sb)

          from  all-shared-acquired-or-owned [OF sharing-consis [OF j-bound jth]] a-in
            all-shared-append [of (takeWhile (Not ∘ is-volatile-Write_sb) sb_j)
            (dropWhile (Not ∘ is-volatile-Write_sb) sb_j)]
          have a-in: a ∈ 𝒪_j ∪ all-acquired sb_j
            by auto
          with ownership-distinct [OF i-bound j-bound neq  ts_sb-i jth]
          have a ∉ (𝒪_sb ∪ all-acquired sb) by auto


          with A-dist R-owned A-R A-shared-owned L-subset a-in
          obtain a ∉ R and a ∉ L
            by fastforce
          then show ?thesis by auto
        qed
      }
      then
      show ?thesis
        apply −
        apply (rule release-all-shared-exchange)
        apply auto
        done
```

qed
}
note release-commute = this
from ownership-distinct-axioms **have** ownership-distinct ts_{sb}.
from sharing-consis-axioms **have** sharing-consis $\mathcal{S}_{sb}$ ts_{sb}.
note share-commute = share-all-until-volatile-write-$Ghost_{sb}$-commute [OF ‹ownership-distinct ts_{sb}›
‹sharing-consis $\mathcal{S}_{sb}$ ts_{sb}› i-bound ts_{sb}-i [simplified sb $Ghost_{sb}$] dist-R-L-A]

have (ts_{sb} [i := (p_{sb},is_{sb}, j_{sb}, sb', $\mathcal{D}_{sb}$, $\mathcal{O}_{sb}$ ∪ A − R,augment-rels (dom $\mathcal{S}_{sb}$) R $\mathcal{R}_{sb}$)],m_{sb},$\mathcal{S}_{sb}'$) ∼ (ts,m,$\mathcal{S}$)
apply (rule sim-config.intros)
apply (simp add: m flush-commute)
apply (clarsimp simp add: $\mathcal{S}$ $\mathcal{S}_{sb}'$ share-commute)
using leq
apply simp
using i-bound i-bound' ts-sim ts-i is-sim $\mathcal{D}$
apply (clarsimp simp add: Let-def nth-list-update sb suspends $Ghost_{sb}$ $\mathcal{R}_{sb}'$ $\mathcal{S}_{sb}'$
split: if-split-asm)
apply (rule conjI)
apply fastforce
apply clarsimp
apply (frule (2) release-commute)
apply clarsimp
apply auto
done
ultimately
show ?thesis
using valid-own' valid-hist' valid-reads' valid-sharing' tmps-distinct'
valid-dd' valid-sops' load-tmps-fresh' enough-flushs'
valid-program-history' valid'
m_{sb}' $\mathcal{S}_{sb}'$ ts_{sb}'
by (auto simp del: fun-upd-apply simp add: $\mathcal{O}_{sb}'$ $\mathcal{R}_{sb}'$)
qed
next
case (Program i p_{sb} is_{sb} j_{sb} sb $\mathcal{D}_{sb}$ $\mathcal{O}_{sb}$ $\mathcal{R}_{sb}$ p_{sb}' mis)
then obtain
ts_{sb}': ts_{sb}' = ts_{sb}[i := (p_{sb}', is_{sb}@mis, j_{sb}, sb@[$Prog_{sb}$ p_{sb} p_{sb}' mis], $\mathcal{D}_{sb}$, $\mathcal{O}_{sb}$,$\mathcal{R}_{sb}$)]
and
i-bound: i < length ts_{sb} **and**
ts_{sb}-i: ts_{sb} ! i = (p_{sb}, is_{sb},j_{sb},sb, $\mathcal{D}_{sb}$, $\mathcal{O}_{sb}$,$\mathcal{R}_{sb}$) **and**
prog: j_{sb} ⊢ p_{sb} $\rightarrow_p$ (p_{sb}',mis) **and**
$\mathcal{S}_{sb}'$: $\mathcal{S}_{sb}'$=$\mathcal{S}_{sb}$ **and**
m_{sb}': m_{sb}'=m_{sb}
by auto

from sim **obtain**
m: m = flush-all-until-volatile-write ts_{sb} m_{sb} **and**
$\mathcal{S}$: $\mathcal{S}$ = share-all-until-volatile-write ts_{sb} $\mathcal{S}_{sb}$ **and**
leq: length ts_{sb} = length ts **and**

ts-sim: $\forall$ i<length ts_{sb}.
  let (p, is_{sb}, j, sb, $\mathcal{D}_{sb}$, $\mathcal{O}_{sb}$,$\mathcal{R}$) = ts_{sb} ! i;
    suspends = dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb
  in $\exists$ is $\mathcal{D}$. instrs suspends @ is_{sb} = is @ prog-instrs suspends $\wedge$
      $\mathcal{D}_{sb}$ = ($\mathcal{D}$ $\vee$ outstanding-refs is-volatile-$Write_{sb}$ sb $\neq$ {}) $\wedge$
    ts ! i =
      (hd-prog p suspends,
      is,
      j |` (dom j − read-tmps suspends), (),
      $\mathcal{D}$,
      acquired True (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb) $\mathcal{O}_{sb}$,
      release (takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb) (dom $\mathcal{S}_{sb}$) $\mathcal{R}$)
  by cases blast

from i-bound leq **have** i-bound$'$: i < length ts
  by auto

have split-sb: sb = takeWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb @ dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb
  (**is** sb = ?take-sb@?drop-sb)
  by simp

from ts-sim [rule-format, OF i-bound] ts_{sb}-i **obtain** suspends is $\mathcal{D}$ **where**
  suspends: suspends = dropWhile (Not $\circ$ is-volatile-$Write_{sb}$) sb **and**
  is-sim: instrs suspends @ is_{sb} = is @ prog-instrs suspends **and**
  $\mathcal{D}$: $\mathcal{D}_{sb}$ = ($\mathcal{D}$ $\vee$ outstanding-refs is-volatile-$Write_{sb}$ sb $\neq$ {}) **and**
  ts-i: ts ! i =
    (hd-prog p_{sb} suspends, is,
    j_{sb} |` (dom j_{sb} − read-tmps suspends), (), $\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$,
    release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
  by (auto simp add: Let-def)

from prog-step-preserves-valid [OF i-bound ts_{sb}-i prog valid]
have valid$'$: valid ts_{sb}'
  by (simp add: ts_{sb}')

have valid-own$'$: valid-ownership $\mathcal{S}_{sb}'$ ts_{sb}'
proof (intro-locales)
  show outstanding-non-volatile-refs-owned-or-read-only $\mathcal{S}_{sb}'$ ts_{sb}'
  proof −
from outstanding-non-volatile-refs-owned-or-read-only [OF i-bound ts_{sb}-i]
have non-volatile-owned-or-read-only False $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ (sb@[$Prog_{sb}$ p_{sb} p_{sb}' mis])
  by (auto simp add: non-volatile-owned-or-read-only-append)
from outstanding-non-volatile-refs-owned-or-read-only-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}' $\mathcal{S}_{sb}'$)
  qed
next
  show outstanding-volatile-writes-unowned-by-others ts_{sb}'
  proof −
have out: outstanding-refs is-volatile-$Write_{sb}$ (sb@[$Prog_{sb}$ p_{sb} p_{sb}' mis]) $\subseteq$

outstanding-refs is-volatile-Write$_{sb}$ sb
by (auto simp add: outstanding-refs-conv)
from outstanding-volatile-writes-unowned-by-others-store-buffer
[OF i-bound ts$_{sb}$-i this]
show ?thesis **by** (simp add: ts$_{sb}$' all-acquired-append)
qed
next
show read-only-reads-unowned ts$_{sb}$'
proof −
have ro: read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) (sb@[Prog$_{sb}$ p$_{sb}$ p$_{sb}$' mis])) $\mathcal{O}_{sb}$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) (sb@[Prog$_{sb}$ p$_{sb}$ p$_{sb}$' mis]))
⊆ read-only-reads (acquired True (takeWhile (Not ∘ is-volatile-Write$_{sb}$) sb) $\mathcal{O}_{sb}$)
(dropWhile (Not ∘ is-volatile-Write$_{sb}$) sb)
apply (case-tac outstanding-refs (is-volatile-Write$_{sb}$) sb = {})
apply (simp-all add: outstanding-vol-write-take-drop-appends
acquired-append read-only-reads-append)
done
have $\mathcal{O}_{sb}$ ∪ all-acquired (sb@[Prog$_{sb}$ p$_{sb}$ p$_{sb}$' mis]) ⊆ $\mathcal{O}_{sb}$ ∪ all-acquired sb
by (auto simp add: all-acquired-append)
from read-only-reads-unowned-nth-update [OF i-bound ts$_{sb}$-i ro this]
show ?thesis
by (simp add: ts$_{sb}$')
qed
next
show ownership-distinct ts$_{sb}$'
proof −
from ownership-distinct-instructions-read-value-store-buffer-independent
[OF i-bound ts$_{sb}$-i, **where** sb'=(sb@[Prog$_{sb}$ p$_{sb}$ p$_{sb}$' mis])]
show ?thesis **by** (simp add: ts$_{sb}$' all-acquired-append)
qed
qed

from valid-last-prog [OF i-bound ts$_{sb}$-i]
have last-prog: last-prog p$_{sb}$ sb = p$_{sb}$**.**

have valid-hist': valid-history program-step ts$_{sb}$'
proof −
from valid-history [OF i-bound ts$_{sb}$-i]
have history-consistent j$_{sb}$ (hd-prog p$_{sb}$ sb) sb**.**
from history-consistent-append-Prog$_{sb}$ [OF prog this last-prog]
have hist-consis': history-consistent j$_{sb}$ (hd-prog p$_{sb}$' (sb@[Prog$_{sb}$ p$_{sb}$ p$_{sb}$' mis]))
(sb@[Prog$_{sb}$ p$_{sb}$ p$_{sb}$' mis])**.**
from valid-history-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts$_{sb}$')
qed

have valid-reads': valid-reads m$_{sb}$ ts$_{sb}$'
proof −

from valid-reads [OF i-bound ts_{sb}-i]
have reads-consistent False $\mathcal{O}_{sb}$ m_{sb} sb **.**
from reads-consistent-snoc-$Prog_{sb}$ [OF this]
have reads-consistent False $\mathcal{O}_{sb}$ m_{sb} (sb@[$Prog_{sb}$ p_{sb} p_{sb}' mis])**.**
from valid-reads-nth-update [OF i-bound this]
show ?thesis **by** (simp add: ts_{sb}')
qed

have valid-sharing$'$: valid-sharing $\mathcal{S}_{sb}'$ ts_{sb}'
proof (intro-locales)
from outstanding-non-volatile-writes-unshared [OF i-bound ts_{sb}-i]
have non-volatile-writes-unshared $\mathcal{S}_{sb}$ (sb@[$Prog_{sb}$ p_{sb} p_{sb}' mis])
by (auto simp add: non-volatile-writes-unshared-append)
from outstanding-non-volatile-writes-unshared-nth-update [OF i-bound this]
show outstanding-non-volatile-writes-unshared $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: ts_{sb}' $\mathcal{S}_{sb}'$)
next
from sharing-consis [OF i-bound ts_{sb}-i]
have sharing-consistent $\mathcal{S}_{sb}$ $\mathcal{O}_{sb}$ (sb@[$Prog_{sb}$ p_{sb} p_{sb}' mis])
by (auto simp add: sharing-consistent-append)
from sharing-consis-nth-update [OF i-bound this]
show sharing-consis $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: ts_{sb}' $\mathcal{S}_{sb}'$)
next
from read-only-unowned-nth-update [OF i-bound read-only-unowned [OF i-bound ts_{sb}-i]]
show read-only-unowned $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: $\mathcal{S}_{sb}'$ ts_{sb}')
next
from unowned-shared-nth-update [OF i-bound ts_{sb}-i subset-refl]
show unowned-shared $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: ts_{sb}' $\mathcal{S}_{sb}'$)
next
from no-outstanding-write-to-read-only-memory [OF i-bound ts_{sb}-i]

have no-write-to-read-only-memory $\mathcal{S}_{sb}$ (sb @ [$Prog_{sb}$ p_{sb} p_{sb}' mis])
by (simp add: no-write-to-read-only-memory-append)

from no-outstanding-write-to-read-only-memory-nth-update [OF i-bound this]
show no-outstanding-write-to-read-only-memory $\mathcal{S}_{sb}'$ ts_{sb}'
by (simp add: $\mathcal{S}_{sb}'$ ts_{sb}')
qed

have tmps-distinct$'$: tmps-distinct ts_{sb}'
proof (intro-locales)
from load-tmps-distinct [OF i-bound ts_{sb}-i]
have distinct-load-tmps is_{sb}**.**
with distinct-load-tmps-prog-step [OF i-bound ts_{sb}-i prog valid]
have distinct-load-tmps (is_{sb}@mis)
by (auto simp add: distinct-load-tmps-append)

from load-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-distinct ts_{sb}'
by (simp add: ts_{sb}')
next
from read-tmps-distinct [OF i-bound ts_{sb}-i]
have distinct-read-tmps (sb@[$Prog_{sb}$ p_{sb} p_{sb}' mis])
by (simp add: distinct-read-tmps-append)
from read-tmps-distinct-nth-update [OF i-bound this]
show read-tmps-distinct ts_{sb}'
by (simp add: ts_{sb}')
next
from load-tmps-read-tmps-distinct [OF i-bound ts_{sb}-i]
distinct-load-tmps-prog-step [OF i-bound ts_{sb}-i prog valid]
have load-tmps (is_{sb}@mis) $\cap$ read-tmps (sb@[$Prog_{sb}$ p_{sb} p_{sb}' mis]) = {}
by (auto simp add: read-tmps-append load-tmps-append)
from load-tmps-read-tmps-distinct-nth-update [OF i-bound this]
show load-tmps-read-tmps-distinct ts_{sb}' **by** (simp add: ts_{sb}')
qed

have valid-dd′: valid-data-dependency ts_{sb}'
proof −
from data-dependency-consistent-instrs [OF i-bound ts_{sb}-i]
have data-dependency-consistent-instrs (dom j_{sb}) is_{sb}.
with valid-data-dependency-prog-step [OF i-bound ts_{sb}-i prog valid]
load-tmps-write-tmps-distinct [OF i-bound ts_{sb}-i]
obtain
data-dependency-consistent-instrs (dom j_{sb}) (is_{sb}@mis)
load-tmps (is_{sb}@mis) $\cap$ $\bigcup$(fst ‘ write-sops (sb@[$Prog_{sb}$ p_{sb} p_{sb}' mis])) = {}
by (force simp add: load-tmps-append data-dependency-consistent-instrs-append write-sops-append)
from valid-data-dependency-nth-update [OF i-bound this]
show ?thesis
by (simp add: ts_{sb}')
qed

have load-tmps-fresh′: load-tmps-fresh ts_{sb}'
proof −

from load-tmps-fresh [OF i-bound ts_{sb}-i]
load-tmps-fresh-prog-step [OF i-bound ts_{sb}-i prog valid]
have load-tmps (is_{sb}@mis) $\cap$ dom j_{sb} = {}
by (auto simp add: load-tmps-append)
from load-tmps-fresh-nth-update [OF i-bound this]
show ?thesis
by (simp add: ts_{sb}')
qed

have enough-flushs′: enough-flushs ts_{sb}'
proof −

from clean-no-outstanding-volatile-Write$_{sb}$ [OF i-bound ts$_{sb}$-i]
have $\neg$ $\mathcal{D}_{sb}$ $\longrightarrow$ outstanding-refs is-volatile-Write$_{sb}$ (sb@[Prog$_{sb}$ p$_{sb}$ p$_{sb}'$ mis]) = {}
by (auto simp add: outstanding-refs-append)

from enough-flushs-nth-update [OF i-bound this]
show ?thesis
by (simp add: ts$_{sb}'$)
qed

have valid-sops$'$: valid-sops ts$_{sb}'$
proof –
from valid-store-sops [OF i-bound ts$_{sb}$-i] valid-sops-prog-step [OF prog]
valid-implies-valid-prog[OF i-bound ts$_{sb}$-i valid]
have valid-store: $\forall$ sop$\in$store-sops (is$_{sb}$@mis). valid-sop sop
by (auto simp add: store-sops-append)

from valid-write-sops [OF i-bound ts$_{sb}$-i]
have $\forall$ sop$\in$write-sops (sb@[Prog$_{sb}$ p$_{sb}$ p$_{sb}'$ mis]). valid-sop sop
by (auto simp add: write-sops-append)
from valid-sops-nth-update [OF i-bound this valid-store]
show ?thesis
by (simp add: ts$_{sb}'$)
qed

have valid-program-history$'$:valid-program-history ts$_{sb}'$
proof –
from valid-program-history [OF i-bound ts$_{sb}$-i]
have causal-program-history is$_{sb}$ sb .
from causal-program-history-Prog$_{sb}$ [OF this]
have causal$'$: causal-program-history (is$_{sb}$@mis) (sb@[Prog$_{sb}$ p$_{sb}$ p$_{sb}'$ mis]).
from last-prog-append-Prog$_{sb}$
have last-prog p$_{sb}'$ (sb@[Prog$_{sb}$ p$_{sb}$ p$_{sb}'$ mis]) = p$_{sb}'$.
from valid-program-history-nth-update [OF i-bound causal$'$ this]
show ?thesis
by (simp add: ts$_{sb}'$)
qed

show ?thesis
proof (cases outstanding-refs is-volatile-Write$_{sb}$ sb = {})
case True
from True **have** flush-all: takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb = sb
by (auto simp add: outstanding-refs-conv)

from True **have** suspend-nothing: dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb = []
by (auto simp add: outstanding-refs-conv)

hence suspends-empty: suspends = []
by (simp add: suspends)

from suspends-empty is-sim **have** is: is = is$_{sb}$

by (simp)

from ts-i **have** ts-i: ts ! i = (p_{sb}, is_{sb}, j_{sb}, (),
$\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$,release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)
by (simp add: suspends-empty is)

from direct-computation.Program [OF i-bound′ ts-i prog]
have (ts,m,$\mathcal{S}$) $\Rightarrow_d$ (ts[i := (p_{sb}′, is_{sb} @ mis, j_{sb}, (),
$\mathcal{D}$, acquired True ?take-sb $\mathcal{O}_{sb}$,release ?take-sb (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)], m, $\mathcal{S}$).

moreover

note flush-commute = flush-all-until-volatile-write-append-$Prog_{sb}$-commute [OF i-bound ts_{sb}-i]

from True
have suspend-nothing′:
(dropWhile (Not ∘ is-volatile-$Write_{sb}$) (sb @ [$Prog_{sb}$ p_{sb} p_{sb}′ mis])) = []
by (auto simp add: outstanding-refs-conv)

note share-commute =
share-all-until-volatile-write-update-sb [OF share-append-$Prog_{sb}$ i-bound ts_{sb}-i]

from $\mathcal{D}$
have $\mathcal{D}$′: $\mathcal{D}_{sb}$ = ($\mathcal{D}$ ∨ outstanding-refs is-volatile-$Write_{sb}$ (sb@[$Prog_{sb}$ p_{sb} p_{sb}′ mis]) ≠ {})
by (auto simp: outstanding-refs-append)

have (ts_{sb} [i := (p_{sb}′,is_{sb}@mis, j_{sb}, sb@[$Prog_{sb}$ p_{sb} p_{sb}′ mis], $\mathcal{D}_{sb}$, $\mathcal{O}_{sb}$,$\mathcal{R}_{sb}$)],
m_{sb},$\mathcal{S}_{sb}$′) ∼
(ts[i:=(p_{sb}′, is_{sb} @ mis, j_{sb}, (), $\mathcal{D}$,
acquired True (takeWhile (Not ∘ is-volatile-$Write_{sb}$)
(sb@[$Prog_{sb}$ p_{sb} p_{sb}′ mis])) $\mathcal{O}_{sb}$,
release (sb@[$Prog_{sb}$ p_{sb} p_{sb}′ mis]) (dom $\mathcal{S}_{sb}$) $\mathcal{R}_{sb}$)],m,$\mathcal{S}$)
apply (rule sim-config.intros)
apply (simp add: m flush-commute)
apply (clarsimp simp add: $\mathcal{S}$ $\mathcal{S}_{sb}$′ share-commute)
using leq
apply simp

using i-bound i-bound′ ts-sim ts-i $\mathcal{D}$′
apply (clarsimp simp add: Let-def nth-list-update flush-all suspend-nothing′ $Prog_{sb}$ $\mathcal{S}_{sb}$′

release-append-$Prog_{sb}$ release-append
split: if-split-asm)
done
ultimately show ?thesis
using valid-own′ valid-hist′ valid-reads′ valid-sharing′ tmps-distinct′ m_{sb}′
valid-dd′ valid-sops′ load-tmps-fresh′ enough-flushs′ valid-sharing′
valid-program-history′ valid′

$\mathcal{S}_{sb}'$ ts_{sb}'

by (auto simp del: fun-upd-apply simp add: acquired-append-Prog$_{sb}$ release-append-Prog$_{sb}$ release-append flush-all)

next

case False

then obtain r **where** r-in: r $\in$ set sb **and** volatile-r: is-volatile-Write$_{sb}$ r

by (auto simp add: outstanding-refs-conv)

from takeWhile-dropWhile-real-prefix

[OF r-in, of (Not $\circ$ is-volatile-Write$_{sb}$), simplified, OF volatile-r]

obtain a$'$ v$'$ sb$''$ sop$'$ A$'$ L$'$ R$'$ W$'$ **where**

sb-split: sb = takeWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb @ Write$_{sb}$ True a$'$ sop$'$ v$'$ A$'$ L$'$ R$'$ W$'$# sb$''$

and

drop: dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb = Write$_{sb}$ True a$'$ sop$'$ v$'$ A$'$ L$'$ R$'$ W$'$# sb$''$

apply (auto)

subgoal for y

apply (case-tac y)

apply auto

done

done

from drop suspends **have** suspends$'$: suspends = Write$_{sb}$ True a$'$ sop$'$ v$'$ A$'$ L$'$ R$'$ W$'$# sb$''$

by simp

have (ts, m, $\mathcal{S}$) $\Rightarrow_d^*$ (ts, m, $\mathcal{S}$) **by** auto

moreover

note flush-commute= flush-all-until-volatile-write-append-Prog$_{sb}$-commute [OF i-bound ts$_{sb}$-i]

have Write$_{sb}$ True a$'$ sop$'$ v$'$ A$'$ L$'$ R$'$ W$'$ $\in$ set sb

by (subst sb-split) auto

from dropWhile-append1 [OF this, of (Not $\circ$ is-volatile-Write$_{sb}$)]

have drop-app-comm:

(dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) (sb @ [Prog$_{sb}$ p$_{sb}$ p$_{sb}'$ mis])) =

dropWhile (Not $\circ$ is-volatile-Write$_{sb}$) sb @ [Prog$_{sb}$ p$_{sb}$ p$_{sb}'$ mis]

by simp

note share-commute =

share-all-until-volatile-write-update-sb [OF share-append-Prog$_{sb}$ i-bound ts$_{sb}$-i]

from $\mathcal{D}$

have $\mathcal{D}'$: $\mathcal{D}_{sb}$ = ($\mathcal{D}$ $\vee$ outstanding-refs is-volatile-Write$_{sb}$ (sb@[Prog$_{sb}$ p$_{sb}$ p$_{sb}'$ mis]) $\neq$ {})

by (auto simp: outstanding-refs-append)

have (ts$_{sb}$ [i := (p$_{sb}'$,is$_{sb}$@mis,j$_{sb}$, sb@[Prog$_{sb}$ p$_{sb}$ p$_{sb}'$ mis], $\mathcal{D}_{sb}$, $\mathcal{O}_{sb}$,$\mathcal{R}_{sb}$)],

$\mathrm{m_{sb}}$,$\mathcal{S}_{\mathsf{sb}}$′) ∼
(ts,m,$\mathcal{S}$)
apply (rule sim-config.intros)
apply (simp add: m flush-commute)
apply (clarsimp simp add: $\mathcal{S}$ $\mathcal{S}_{\mathsf{sb}}$′ share-commute)
using leq
apply simp

using i-bound i-bound′ ts-sim ts-i is-sim suspends suspends′ [simplified suspends] $\mathcal{D}$′
apply (clarsimp simp add: Let-def nth-list-update $\mathrm{Prog_{sb}}$
drop-app-comm instrs-append prog-instrs-append
read-tmps-append hd-prog-append-$\mathrm{Prog_{sb}}$ acquired-append-$\mathrm{Prog_{sb}}$ release-append-$\mathrm{Prog_{sb}}$ release-append $\mathcal{S}_{\mathsf{sb}}$′
split: if-split-asm)
done

ultimately show ?thesis
using valid-own′ valid-hist′ valid-reads′ valid-sharing′ tmps-distinct′ $\mathrm{m_{sb}}$′
valid-dd′ valid-sops′ load-tmps-fresh′ enough-flushs′ valid-sharing′
valid-program-history′ valid′
$\mathcal{S}_{\mathsf{sb}}$′ $\mathrm{ts_{sb}}$′
by (auto simp del: fun-upd-apply)
qed
qed
qed

theorem (**in** xvalid-program) concurrent-direct-steps-simulates-store-buffer-history-steps:
assumes step-sb: ($\mathrm{ts_{sb}}$,$\mathrm{m_{sb}}$,$\mathcal{S}_{\mathsf{sb}}$) $\Rightarrow_{\mathsf{sbh}}{}^*$ ($\mathrm{ts_{sb}}$′,$\mathrm{m_{sb}}$′,$\mathcal{S}_{\mathsf{sb}}$′)
assumes valid-own: valid-ownership $\mathcal{S}_{\mathsf{sb}}$ $\mathrm{ts_{sb}}$
assumes valid-sb-reads: valid-reads $\mathrm{m_{sb}}$ $\mathrm{ts_{sb}}$
assumes valid-hist: valid-history program-step $\mathrm{ts_{sb}}$
assumes valid-sharing: valid-sharing $\mathcal{S}_{\mathsf{sb}}$ $\mathrm{ts_{sb}}$
assumes tmps-distinct: tmps-distinct $\mathrm{ts_{sb}}$
assumes valid-sops: valid-sops $\mathrm{ts_{sb}}$
assumes valid-dd: valid-data-dependency $\mathrm{ts_{sb}}$
assumes load-tmps-fresh: load-tmps-fresh $\mathrm{ts_{sb}}$
assumes enough-flushs: enough-flushs $\mathrm{ts_{sb}}$
assumes valid-program-history: valid-program-history $\mathrm{ts_{sb}}$
assumes valid: valid $\mathrm{ts_{sb}}$
shows $\bigwedge$ts $\mathcal{S}$ m. ($\mathrm{ts_{sb}}$,$\mathrm{m_{sb}}$,$\mathcal{S}_{\mathsf{sb}}$) ∼ (ts,m,$\mathcal{S}$) $\Longrightarrow$ safe-reach-direct safe-delayed (ts,m,$\mathcal{S}$) $\Longrightarrow$
valid-ownership $\mathcal{S}_{\mathsf{sb}}$′ $\mathrm{ts_{sb}}$′ ∧ valid-reads $\mathrm{m_{sb}}$′ $\mathrm{ts_{sb}}$′ ∧ valid-history program-step $\mathrm{ts_{sb}}$′ ∧
valid-sharing $\mathcal{S}_{\mathsf{sb}}$′ $\mathrm{ts_{sb}}$′ ∧ tmps-distinct $\mathrm{ts_{sb}}$′ ∧ valid-data-dependency $\mathrm{ts_{sb}}$′ ∧
valid-sops $\mathrm{ts_{sb}}$′ ∧ load-tmps-fresh $\mathrm{ts_{sb}}$′ ∧ enough-flushs $\mathrm{ts_{sb}}$′ ∧
valid-program-history $\mathrm{ts_{sb}}$′ ∧ valid $\mathrm{ts_{sb}}$′ ∧
(∃ts′ m′ $\mathcal{S}$′. (ts,m,$\mathcal{S}$) $\Rightarrow_{\mathsf{d}}{}^*$ (ts′,m′,$\mathcal{S}$′) ∧ ($\mathrm{ts_{sb}}$′,$\mathrm{m_{sb}}$′,$\mathcal{S}_{\mathsf{sb}}$′) ∼ (ts′,m′,$\mathcal{S}$′))
using step-sb valid-own valid-sb-reads valid-hist valid-sharing tmps-distinct valid-sops
valid-dd load-tmps-fresh enough-flushs valid-program-history valid

proof (induct rule: converse-rtranclp-induct-sbh-steps)
case refl **thus** ?case
by auto
next
case (step ts_{sb} m_{sb} $\mathcal{S}_{sb}$ ts_{sb}'' m_{sb}'' $\mathcal{S}_{sb}''$)
note first = ‹(ts_{sb}, m_{sb}, $\mathcal{S}_{sb}$) $\Rightarrow_{sbh}$ (ts_{sb}'', m_{sb}'', $\mathcal{S}_{sb}''$)›
note sim = ‹(ts_{sb}, m_{sb}, $\mathcal{S}_{sb}$) $\sim$ (ts, m, $\mathcal{S}$)›
note safe-reach = ‹safe-reach-direct safe-delayed (ts, m, $\mathcal{S}$)›
note valid-own = ‹valid-ownership $\mathcal{S}_{sb}$ ts_{sb}›
note valid-reads = ‹valid-reads m_{sb} ts_{sb}›
note valid-hist = ‹valid-history program-step ts_{sb}›
note valid-sharing = ‹valid-sharing $\mathcal{S}_{sb}$ ts_{sb}›
note tmps-distinct = ‹tmps-distinct ts_{sb}›
note valid-sops = ‹valid-sops ts_{sb}›
note valid-dd = ‹valid-data-dependency ts_{sb}›
note load-tmps-fresh = ‹load-tmps-fresh ts_{sb}›
note enough-flushs = ‹enough-flushs ts_{sb}›
note valid-prog-hist = ‹valid-program-history ts_{sb}›
note valid = ‹valid ts_{sb}›
from concurrent-direct-steps-simulates-store-buffer-history-step [OF first
valid-own valid-reads valid-hist valid-sharing tmps-distinct valid-sops valid-dd
load-tmps-fresh enough-flushs valid-prog-hist valid sim safe-reach]
obtain ts″ m″ $\mathcal{S}''$ **where**
valid-own″: valid-ownership $\mathcal{S}_{sb}''$ ts_{sb}'' **and**
valid-reads″: valid-reads m_{sb}'' ts_{sb}'' **and**
valid-hist″: valid-history program-step ts_{sb}'' **and**
valid-sharing″: valid-sharing $\mathcal{S}_{sb}''$ ts_{sb}'' **and**
tmps-dist″: tmps-distinct ts_{sb}'' **and**
valid-dd″: valid-data-dependency ts_{sb}'' **and**
valid-sops″: valid-sops ts_{sb}'' **and**
load-tmps-fresh″: load-tmps-fresh ts_{sb}'' **and**
enough-flushs″: enough-flushs ts_{sb}'' **and**
valid-prog-hist″: valid-program-history ts_{sb}''**and**
valid″: valid ts_{sb}'' **and**
steps: (ts, m, $\mathcal{S}$) $\Rightarrow_d^*$ (ts″, m″, $\mathcal{S}''$) **and**
sim: (ts_{sb}'', m_{sb}'',$\mathcal{S}_{sb}''$) $\sim$ (ts″, m″,$\mathcal{S}''$)
by blast

from step.hyps (3) [OF sim safe-reach-steps [OF safe-reach steps] valid-own″ valid-reads″
valid-hist″ valid-sharing″
tmps-dist″ valid-sops″ valid-dd″ load-tmps-fresh″ enough-flushs″ valid-prog-hist″ valid″
]

obtain ts′ m′ $\mathcal{S}'$ **where**
valid: valid-ownership $\mathcal{S}_{sb}'$ ts_{sb}' valid-reads m_{sb}' ts_{sb}' valid-history program-step ts_{sb}'
valid-sharing $\mathcal{S}_{sb}'$ ts_{sb}' tmps-distinct ts_{sb}' valid-data-dependency ts_{sb}'
valid-sops ts_{sb}' load-tmps-fresh ts_{sb}' enough-flushs ts_{sb}'
valid-program-history ts_{sb}' valid ts_{sb}' **and**
last: (ts″, m″, $\mathcal{S}''$) $\Rightarrow_d^*$ (ts′, m′, $\mathcal{S}'$) **and**

sim′: (ts_{sb}′, m_{sb}′,$\mathcal{S}_{sb}$′) ∼ (ts′, m′,$\mathcal{S}$′)
by blast

note steps **also note** last
finally show ?case
using valid sim′
by blast
qed

sublocale $initial_{sb}$ ⊆ tmps-distinct ..
locale xvalid-program-progress = program-progress + xvalid-program

theorem (**in** xvalid-program-progress) concurrent-direct-execution-simulates-store-buffer-history-execution:
assumes exec-sb: (ts_{sb},m_{sb},$\mathcal{S}_{sb}$) $\Rightarrow_{sbh}{}^*$ (ts_{sb}′,m_{sb}′,$\mathcal{S}_{sb}$′)
assumes init: $initial_{sb}$ ts_{sb} $\mathcal{S}_{sb}$
assumes valid: valid ts_{sb}
assumes sim: (ts_{sb},m_{sb},$\mathcal{S}_{sb}$) ∼ (ts,m,$\mathcal{S}$)
assumes safe: safe-reach-direct safe-free-flowing (ts,m,$\mathcal{S}$)
shows ∃ ts′ m′ $\mathcal{S}$′. (ts,m,$\mathcal{S}$) $\Rightarrow_{d}{}^*$ (ts′,m′,$\mathcal{S}$′) ∧
(ts_{sb}′,m_{sb}′,$\mathcal{S}_{sb}$′) ∼ (ts′,m′,$\mathcal{S}$′)
proof −
from init **interpret** ini: $initial_{sb}$ ts_{sb} $\mathcal{S}_{sb}$.
from safe-free-flowing-implies-safe-delayed′ [OF init sim safe]
have safe-delayed: safe-reach-direct safe-delayed (ts, m, $\mathcal{S}$).
from local.ini.valid-ownership-axioms **have** valid-ownership $\mathcal{S}_{sb}$ ts_{sb} .
from local.ini.valid-reads-axioms **have** valid-reads m_{sb} ts_{sb}.
from local.ini.valid-history-axioms **have** valid-history program-step ts_{sb}.
from local.ini.valid-sharing-axioms **have** valid-sharing $\mathcal{S}_{sb}$ ts_{sb}.
from local.ini.tmps-distinct-axioms **have** tmps-distinct ts_{sb}.
from local.ini.valid-sops-axioms **have** valid-sops ts_{sb}.
from local.ini.valid-data-dependency-axioms **have** valid-data-dependency ts_{sb}.
from local.ini.load-tmps-fresh-axioms **have** load-tmps-fresh ts_{sb}.
from local.ini.enough-flushs-axioms **have** enough-flushs ts_{sb}.
from local.ini.valid-program-history-axioms **have** valid-program-history ts_{sb}.
from concurrent-direct-steps-simulates-store-buffer-history-steps [OF exec-sb
‹valid-ownership $\mathcal{S}_{sb}$ ts_{sb}›
‹valid-reads m_{sb} ts_{sb}› ‹valid-history program-step ts_{sb}›
‹valid-sharing $\mathcal{S}_{sb}$ ts_{sb}› ‹tmps-distinct ts_{sb}› ‹valid-sops ts_{sb}›
‹valid-data-dependency ts_{sb}› ‹load-tmps-fresh ts_{sb}› ‹enough-flushs ts_{sb}›
‹valid-program-history ts_{sb}› valid sim safe-delayed]
show ?thesis **by** auto
qed

lemma filter-is-$Write_{sb}$-Cons-$Write_{sb}$: filter is-$Write_{sb}$ xs = $Write_{sb}$ volatile a sop v A L R W#ys

$\Longrightarrow \exists$ rs rws. ($\forall$ r $\in$ set rs. is-Read$_{sb}$ r $\vee$ is-Prog$_{sb}$ r $\vee$ is-Ghost$_{sb}$ r) $\wedge$
xs=rs@Write$_{sb}$ volatile a sop v A L R W#rws $\wedge$ ys=filter is-Write$_{sb}$ rws
proof (induct xs)
case Nil **thus** ?case **by** simp
next
case (Cons x xs)
note feq = ‹filter is-Write$_{sb}$ (x#xs) = Write$_{sb}$ volatile a sop v A L R W# ys›
show ?case
proof (cases x)
case (Write$_{sb}$ volatile′ a′ sop′ v′ A′ L′ R′ W′)
with feq **obtain** volatile′=volatile a′=a v′=v sop′=sop A′=A L′=L R′=R W′=W
ys = filter is-Write$_{sb}$ xs
by auto
thus ?thesis
apply –
apply (rule-tac x=[] **in** exI)
apply (rule-tac x=xs **in** exI)
apply (simp add: Write$_{sb}$)
done
next
case (Read$_{sb}$ volatile′ a′ t′ v′)
from feq **have** filter is-Write$_{sb}$ xs = Write$_{sb}$ volatile a sop v A L R W#ys
by (simp add: Read$_{sb}$)
from Cons.hyps [OF this] **obtain** rs rws **where**
$\forall$ r $\in$ set rs. is-Read$_{sb}$ r $\vee$ is-Prog$_{sb}$ r $\vee$ is-Ghost$_{sb}$ r **and**
xs=rs @ Write$_{sb}$ volatile a sop v A L R W# rws **and**
ys=filter is-Write$_{sb}$ rws
by clarsimp
then show ?thesis
apply –
apply (rule-tac x=Read$_{sb}$ volatile′ a′ t′ v′#rs **in** exI)
apply (rule-tac x=rws **in** exI)
apply (simp add: Read$_{sb}$)
done
next
case (Prog$_{sb}$ p_1 p_2 mis)
from feq **have** filter is-Write$_{sb}$ xs = Write$_{sb}$ volatile a sop v A L R W#ys
by (simp add: Prog$_{sb}$)
from Cons.hyps [OF this] **obtain** rs rws **where**
$\forall$ r $\in$ set rs. is-Read$_{sb}$ r $\vee$ is-Prog$_{sb}$ r $\vee$ is-Ghost$_{sb}$ r **and**
xs=rs @ Write$_{sb}$ volatile a sop v A L R W# rws **and**
ys=filter is-Write$_{sb}$ rws
by clarsimp
then show ?thesis
apply –
apply (rule-tac x=Prog$_{sb}$ p_1 p_2 mis#rs **in** exI)
apply (rule-tac x=rws **in** exI)
apply (simp add: Prog$_{sb}$)
done
next

case ($\text{Ghost}_{\mathsf{sb}}$ A′ L′ R′ W′)
from feq **have** filter is-$\text{Write}_{\mathsf{sb}}$ xs = $\text{Write}_{\mathsf{sb}}$ volatile a sop v A L R W # ys
by (simp add: $\text{Ghost}_{\mathsf{sb}}$)
from Cons.hyps [OF this] **obtain** rs rws **where**
$\forall$ r $\in$ set rs. is-$\text{Read}_{\mathsf{sb}}$ r $\vee$ is-$\text{Prog}_{\mathsf{sb}}$ r $\vee$ is-$\text{Ghost}_{\mathsf{sb}}$ r **and**
xs=rs @ $\text{Write}_{\mathsf{sb}}$ volatile a sop v A L R W# rws **and**
ys=filter is-$\text{Write}_{\mathsf{sb}}$ rws
by clarsimp
then show ?thesis
apply −
apply (rule-tac x=$\text{Ghost}_{\mathsf{sb}}$ A′ L′ R′ W′#rs **in** exI)
apply (rule-tac x=rws **in** exI)
apply (simp add: $\text{Ghost}_{\mathsf{sb}}$)
done
qed
qed

lemma filter-is-$\text{Write}_{\mathsf{sb}}$-empty: filter is-$\text{Write}_{\mathsf{sb}}$ xs = []
$\Longrightarrow$ ($\forall$ r $\in$ set xs. is-$\text{Read}_{\mathsf{sb}}$ r $\vee$ is-$\text{Prog}_{\mathsf{sb}}$ r $\vee$ is-$\text{Ghost}_{\mathsf{sb}}$ r)
proof (induct xs)
case Nil **thus** ?case **by** simp
next
case (Cons x xs)
note feq = ‹filter is-$\text{Write}_{\mathsf{sb}}$ (x#xs) = []›
show ?case
proof (cases x)
case ($\text{Write}_{\mathsf{sb}}$ volatile′ a′ v′)
with feq **have** False
by simp
thus ?thesis **..**
next
case ($\text{Read}_{\mathsf{sb}}$ a′ v′)
from feq **have** filter is-$\text{Write}_{\mathsf{sb}}$ xs = []
by (simp add: $\text{Read}_{\mathsf{sb}}$)
from Cons.hyps [OF this] **obtain**
$\forall$ r $\in$ set xs. is-$\text{Read}_{\mathsf{sb}}$ r $\vee$ is-$\text{Prog}_{\mathsf{sb}}$ r $\vee$ is-$\text{Ghost}_{\mathsf{sb}}$ r
by clarsimp
then show ?thesis
by (simp add: $\text{Read}_{\mathsf{sb}}$)
next
case ($\text{Prog}_{\mathsf{sb}}$ p_2 p_2 mis)
from feq **have** filter is-$\text{Write}_{\mathsf{sb}}$ xs = []
by (simp add: $\text{Prog}_{\mathsf{sb}}$)
from Cons.hyps [OF this] **obtain**
$\forall$ r $\in$ set xs. is-$\text{Read}_{\mathsf{sb}}$ r $\vee$ is-$\text{Prog}_{\mathsf{sb}}$ r $\vee$ is-$\text{Ghost}_{\mathsf{sb}}$ r
by clarsimp
then show ?thesis
by (simp add: $\text{Prog}_{\mathsf{sb}}$)
next
case ($\text{Ghost}_{\mathsf{sb}}$ A′ L′ R′ W′)

from feq **have** filter is-Write$_{sb}$ xs = []
  by (simp add: Ghost$_{sb}$)
from Cons.hyps [OF this] **obtain**
  $\forall$ r $\in$ set xs. is-Read$_{sb}$ r $\vee$ is-Prog$_{sb}$ r $\vee$ is-Ghost$_{sb}$ r
  by clarsimp
then show ?thesis
  by (simp add: Ghost$_{sb}$)
qed
qed

lemma flush-reads-program: $\bigwedge\mathcal{O}\ \mathcal{S}\ \mathcal{R}$.
$\forall$ r $\in$ set sb. is-Read$_{sb}$ r $\vee$ is-Prog$_{sb}$ r $\vee$ is-Ghost$_{sb}$ r $\Longrightarrow$
$\exists\,\mathcal{O}'\ \mathcal{R}'\ \mathcal{S}'$. (m,sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}{}^*$ (m,[],$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$)
proof (induct sb)
case Nil **thus** ?case **by** auto
next
case (Cons x sb)
note ‹$\forall$ r$\in$set (x # sb). is-Read$_{sb}$ r $\vee$ is-Prog$_{sb}$ r $\vee$ is-Ghost$_{sb}$ r›
then obtain x: is-Read$_{sb}$ x $\vee$ is-Prog$_{sb}$ x $\vee$ is-Ghost$_{sb}$ x **and** sb: $\forall$ r$\in$set sb. is-Read$_{sb}$ r $\vee$ is-Prog$_{sb}$ r $\vee$ is-Ghost$_{sb}$ r
  by (cases x) auto

{
assume is-Read$_{sb}$ x
then obtain volatile a t v **where** x: x=Read$_{sb}$ volatile a t v
  by (cases x) auto

have (m,Read$_{sb}$ volatile a t v#sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}$ (m,sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$)
  by (rule Read$_{sb}$)
also
from Cons.hyps [OF sb] **obtain** $\mathcal{O}'\ \mathcal{S}'$ acq$'$ $\mathcal{R}'$
  where (m, sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}{}^*$ (m, [],$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$) **by** blast
finally
have ?case
  by (auto simp add: x)
}
moreover
{
assume is-Prog$_{sb}$ x
then obtain p_1 p_2 mis **where** x: x=Prog$_{sb}$ p_1 p_2 mis
  by (cases x) auto

have (m,Prog$_{sb}$ p_1 p_2 mis#sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}$ (m,sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$)
  by (rule Prog$_{sb}$)
also
from Cons.hyps [OF sb] **obtain** $\mathcal{O}'\ \mathcal{R}'\ \mathcal{S}'$ acq$'$
where (m, sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}{}^*$ (m, [],$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$) **by** blast
finally
have ?case

by (auto simp add: x)
}
moreover
{
assume is-Ghost$_{sb}$ x
then obtain A L R W **where** x: x=Ghost$_{sb}$ A L R W
by (cases x) auto

have (m,Ghost$_{sb}$ A L R W#sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{f}$ (m,sb,$\mathcal{O}$ $\cup$ A $-$ R,augment-rels (dom $\mathcal{S}$) R $\mathcal{R}$,$\mathcal{S}$ $\oplus_{W}$ R $\ominus_{A}$ L)
by (rule Ghost)
also
from Cons.hyps [OF sb] **obtain** $\mathcal{O}'$ $\mathcal{S}'$ $\mathcal{R}'$ acq$'$
where (m, sb,$\mathcal{O}$ $\cup$ A $-$ R ,augment-rels (dom $\mathcal{S}$) R $\mathcal{R}$,$\mathcal{S}$ $\oplus_{W}$ R $\ominus_{A}$ L) $\rightarrow_{f}^{*}$ (m, [],$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$) **by** blast
finally
have ?case
by (auto simp add: x)
}
ultimately show ?case
using x **by** blast
qed

lemma flush-progress: $\exists$ m$'$ $\mathcal{O}'$ $\mathcal{S}'$ $\mathcal{R}'$. (m,r#sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{f}$ (m$'$,sb,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$)
proof (cases r)
case (Write$_{sb}$ volatile a sop v A L R W)
from flush-step.Write$_{sb}$ [OF refl refl refl, of m volatile a sop v A L R W sb $\mathcal{O}$ $\mathcal{R}$ $\mathcal{S}$]
show ?thesis
by (auto simp add: Write$_{sb}$)
next
case (Read$_{sb}$ volatile a t v)
from flush-step.Read$_{sb}$ [of m volatile a t v sb $\mathcal{O}$ $\mathcal{R}$ $\mathcal{S}$]
show ?thesis
by (auto simp add: Read$_{sb}$)
next
case (Prog$_{sb}$ p$_1$ p$_2$ mis)
from flush-step.Prog$_{sb}$ [of m p$_1$ p$_2$ mis sb $\mathcal{O}$ $\mathcal{R}$ $\mathcal{S}$]
show ?thesis
by (auto simp add: Prog$_{sb}$)
next
case (Ghost$_{sb}$ A L R W)
from flush-step.Ghost [of m A L R W sb $\mathcal{O}$ $\mathcal{R}$ $\mathcal{S}$]
show ?thesis
by (auto simp add: Ghost$_{sb}$)
qed

lemma flush-empty:
assumes steps: (m, sb,$\mathcal{O}$,$\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{f}^{*}$ (m$'$, sb$'$,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$)
shows sb=[] $\Longrightarrow$ m$'$=m $\wedge$ sb$'$=[] $\wedge$ $\mathcal{O}'$=$\mathcal{O}$ $\wedge$ $\mathcal{R}'$=$\mathcal{R}$ $\wedge$ $\mathcal{S}'$=$\mathcal{S}$

using steps
apply (induct rule: converse-rtranclp-induct5)
apply (auto elim: flush-step.cases)
done

lemma flush-append:
assumes steps: (m, sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}{}^*$ (m′, sb′,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$)
shows $\bigwedge$xs. (m, sb@xs,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}{}^*$ (m′, sb′@xs,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$)
using steps
proof (induct rule: converse-rtranclp-induct5)
case refl **thus** ?case **by** auto
next
case (step m sb $\mathcal{O}$ $\mathcal{R}$ $\mathcal{S}$ m″ sb″ $\mathcal{O}''$ $\mathcal{R}''$ $\mathcal{S}''$)
note first= ‹(m,sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}$ (m″,sb″,$\mathcal{O}''$,$\mathcal{R}''$,$\mathcal{S}''$)›
note rest = ‹(m″, sb″,$\mathcal{O}''$,$\mathcal{R}''$,$\mathcal{S}''$) $\rightarrow_{\mathsf{f}}{}^*$ (m′, sb′,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$)›
from step.hyps (3) **have** append-rest: (m″, sb″@xs,$\mathcal{O}''$,$\mathcal{R}''$,$\mathcal{S}''$) $\rightarrow_{\mathsf{f}}{}^*$ (m′, sb′@xs,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$)**.**
from first **show** ?case
proof (cases)
case (Write$_{\mathsf{sb}}$ volatile A R W L a sop v)
then obtain sb: sb=Write$_{\mathsf{sb}}$ volatile a sop v A L R W#sb″ **and** m″: m″=m(a:=v) **and**
$\mathcal{O}''$: $\mathcal{O}''$=(if volatile then $\mathcal{O}$ ∪ A − R else $\mathcal{O}$) **and**
$\mathcal{R}''$: $\mathcal{R}''$=(if volatile then Map.empty else $\mathcal{R}$) **and**
$\mathcal{S}''$: $\mathcal{S}''$=(if volatile then $\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L else $\mathcal{S}$)
by auto
have (m,Write$_{\mathsf{sb}}$ volatile a sop v A L R W#sb″@xs,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}$
(m(a:=v),sb″@xs,if volatile then $\mathcal{O}$ ∪ A − R else $\mathcal{O}$,if volatile then Map.empty else $\mathcal{R}$,
if volatile then $\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L else $\mathcal{S}$)
apply (rule flush-step.Write$_{\mathsf{sb}}$)
apply auto
done
hence (m,sb@xs,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}$ (m″,sb″@xs,$\mathcal{O}''$,$\mathcal{R}''$,$\mathcal{S}''$)
by (simp add: sb m″ $\mathcal{O}''$ $\mathcal{R}''$ $\mathcal{S}''$)
also note append-rest
finally show ?thesis **.**
next
case (Read$_{\mathsf{sb}}$ volatile a t v)
then obtain sb: sb=Read$_{\mathsf{sb}}$ volatile a t v #sb″ **and** m″: m″=m
and $\mathcal{O}''$: $\mathcal{O}''$=$\mathcal{O}$ **and** $\mathcal{S}''$: $\mathcal{S}''$=$\mathcal{S}$ **and** $\mathcal{R}''$: $\mathcal{R}''$=$\mathcal{R}$
by auto
have (m,Read$_{\mathsf{sb}}$ volatile a t v#sb″@xs,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}$ (m,sb″@xs,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$)
by (rule flush-step.Read$_{\mathsf{sb}}$)
hence (m,sb@xs,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}$ (m″,sb″@xs,$\mathcal{O}''$,$\mathcal{R}''$,$\mathcal{S}''$)
by (simp add: sb m″ $\mathcal{O}''$ $\mathcal{R}''$ $\mathcal{S}''$)
also note append-rest
finally show ?thesis **.**
next
case (Prog$_{\mathsf{sb}}$ p_1 p_2 mis)

then obtain sb: sb=Prog$_{\mathsf{sb}}$ p$_1$ p$_2$ mis#sb$''$ **and** m$''$: m$''$=m
and $\mathcal{O}''$: $\mathcal{O}''$=$\mathcal{O}$ **and** $\mathcal{S}''$: $\mathcal{S}''$=$\mathcal{S}$ **and** $\mathcal{R}''$: $\mathcal{R}''$=$\mathcal{R}$
by auto
have (m,Prog$_{\mathsf{sb}}$ p$_1$ p$_2$ mis#sb$''$@xs,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}$ (m,sb$''$@xs,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$)
by (rule flush-step.Prog$_{\mathsf{sb}}$)
hence (m,sb@xs,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}$ (m$''$,sb$''$@xs,$\mathcal{O}''$,$\mathcal{R}''$,$\mathcal{S}''$)
by (simp add: sb m$''$ $\mathcal{O}''$ $\mathcal{R}''$ $\mathcal{S}''$)
also note append-rest
finally show ?thesis **.**
next
case (Ghost A L R W)
then obtain sb: sb=Ghost$_{\mathsf{sb}}$ A L R W#sb$''$ **and** m$''$: m$''$=m
and $\mathcal{O}''$: $\mathcal{O}''$=$\mathcal{O}$ $\cup$ A $-$ R **and** $\mathcal{S}''$: $\mathcal{S}''$=$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L **and**
$\mathcal{R}''$: $\mathcal{R}''$=augment-rels (dom $\mathcal{S}$) R $\mathcal{R}$
by auto
have (m,Ghost$_{\mathsf{sb}}$ A L R W#sb$''$@xs,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}$ (m,sb$''$@xs,$\mathcal{O}$ $\cup$ A $-$ R,augment-rels
(dom $\mathcal{S}$) R $\mathcal{R}$,$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (rule flush-step.Ghost)
hence (m,sb@xs,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}$ (m$''$,sb$''$@xs,$\mathcal{O}''$,$\mathcal{R}''$,$\mathcal{S}''$)
by (simp add: sb m$''$ $\mathcal{O}''$ $\mathcal{R}''$ $\mathcal{S}''$)
also note append-rest
finally show ?thesis **.**
qed
qed

lemmas store-buffer-step-induct =
store-buffer-step.induct [split-format (complete),
consumes 1, case-names SBWrite$_{\mathsf{sb}}$]
theorem flush-simulates-filter-writes:
assumes step: (m,sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{w}}$ (m$'$,sb$'$,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$)
shows $\bigwedge$sb$_{\mathsf{h}}$ $\mathcal{O}_{\mathsf{h}}$ $\mathcal{R}_{\mathsf{h}}$ $\mathcal{S}_{\mathsf{h}}$. sb=filter is-Write$_{\mathsf{sb}}$ sb$_{\mathsf{h}}$
$\Longrightarrow$
$\exists$sb$_{\mathsf{h}}'$ $\mathcal{O}_{\mathsf{h}}'$ $\mathcal{R}_{\mathsf{h}}'$ $\mathcal{S}_{\mathsf{h}}'$. (m,sb$_{\mathsf{h}}$,$\mathcal{O}_{\mathsf{h}}$,$\mathcal{R}_{\mathsf{h}}$,$\mathcal{S}_{\mathsf{h}}$) $\rightarrow_{\mathsf{f}}^*$ (m$'$,sb$_{\mathsf{h}}'$,$\mathcal{O}_{\mathsf{h}}'$,$\mathcal{R}_{\mathsf{h}}'$,$\mathcal{S}_{\mathsf{h}}'$) $\wedge$
sb$'$=filter is-Write$_{\mathsf{sb}}$ sb$_{\mathsf{h}}'$ $\wedge$ (sb$'$=[] $\longrightarrow$ sb$_{\mathsf{h}}'$=[])
using step
proof (induct rule: store-buffer-step-induct)
case (SBWrite$_{\mathsf{sb}}$ m volatile a D f v A L R W sb $\mathcal{O}$ $\mathcal{R}$ $\mathcal{S}$)
note filter-Write$_{\mathsf{sb}}$ = ‹Write$_{\mathsf{sb}}$ volatile a (D,f) v A L R W# sb = filter is-Write$_{\mathsf{sb}}$ sb$_{\mathsf{h}}$›

from filter-is-Write$_{\mathsf{sb}}$-Cons-Write$_{\mathsf{sb}}$ [OF filter-Write$_{\mathsf{sb}}$ [symmetric]]
obtain rs rws **where**
rs-reads: $\forall$r$\in$set rs. is-Read$_{\mathsf{sb}}$ r $\vee$ is-Prog$_{\mathsf{sb}}$ r $\vee$ is-Ghost$_{\mathsf{sb}}$ r **and**
sb$_{\mathsf{h}}$: sb$_{\mathsf{h}}$ = rs @ Write$_{\mathsf{sb}}$ volatile a (D,f) v A L R W# rws **and**
sb: sb = filter is-Write$_{\mathsf{sb}}$ rws
by blast

from flush-reads-program [OF rs-reads] **obtain** $\mathcal{O}_{\mathsf{h}}'$ $\mathcal{R}_{\mathsf{h}}'$ $\mathcal{S}_{\mathsf{h}}'$ acq$_{\mathsf{h}}'$
where (m, rs,$\mathcal{O}_{\mathsf{h}}$,$\mathcal{R}_{\mathsf{h}}$,$\mathcal{S}_{\mathsf{h}}$) $\rightarrow_{\mathsf{f}}^*$ (m, [],$\mathcal{O}_{\mathsf{h}}'$,$\mathcal{R}_{\mathsf{h}}'$,$\mathcal{S}_{\mathsf{h}}'$) **by** blast
from flush-append [OF this]

have (m, rs@$\text{Write}_{\mathsf{sb}}$ volatile a (D,f) v A L R W# rws,$\mathcal{O}_{\mathsf{h}}$,$\mathcal{R}_{\mathsf{h}}$,$\mathcal{S}_{\mathsf{h}}$) $\rightarrow_{\mathsf{f}}^*$
(m, $\text{Write}_{\mathsf{sb}}$ volatile a (D,f) v A L R W# rws,$\mathcal{O}_{\mathsf{h}}'$,$\mathcal{R}_{\mathsf{h}}'$,$\mathcal{S}_{\mathsf{h}}'$)
by simp
also
from flush-step.$\text{Write}_{\mathsf{sb}}$ [OF refl refl refl, of m volatile a (D,f) v A L R W rws $\mathcal{O}_{\mathsf{h}}'$ $\mathcal{R}_{\mathsf{h}}'$ $\mathcal{S}_{\mathsf{h}}'$]
obtain $\mathcal{O}_{\mathsf{h}}''$ $\mathcal{R}_{\mathsf{h}}''$ $\mathcal{S}_{\mathsf{h}}''$
where (m, $\text{Write}_{\mathsf{sb}}$ volatile a (D,f) v A L R W# rws,$\mathcal{O}_{\mathsf{h}}'$,$\mathcal{R}_{\mathsf{h}}'$,$\mathcal{S}_{\mathsf{h}}'$) $\rightarrow_{\mathsf{f}}$ (m(a:=v), rws, $\mathcal{O}_{\mathsf{h}}''$,$\mathcal{R}_{\mathsf{h}}''$,$\mathcal{S}_{\mathsf{h}}''$)
by auto
finally have steps: (m, sb_{h},$\mathcal{O}_{\mathsf{h}}$,$\mathcal{R}_{\mathsf{h}}$,$\mathcal{S}_{\mathsf{h}}$) $\rightarrow_{\mathsf{f}}^*$ (m(a:=v), rws,$\mathcal{O}_{\mathsf{h}}''$,$\mathcal{R}_{\mathsf{h}}''$,$\mathcal{S}_{\mathsf{h}}''$)
by (simp add: sb_{h} sb)
show ?case
proof (cases sb)
case Cons
with steps sb **show** ?thesis
by fastforce
next
case Nil
from filter-is-$\text{Write}_{\mathsf{sb}}$-empty [OF sb [simplified Nil, symmetric]]
have $\forall$r$\in$set rws. is-$\text{Read}_{\mathsf{sb}}$ r $\vee$ is-$\text{Prog}_{\mathsf{sb}}$ r $\vee$ is-$\text{Ghost}_{\mathsf{sb}}$ r.
from flush-reads-program [OF this] **obtain** $\mathcal{O}_{\mathsf{h}}'''$ $\mathcal{R}_{\mathsf{h}}'''$ $\mathcal{S}_{\mathsf{h}}'''$ $\text{acq}_{\mathsf{h}}'''$
where (m(a:=v), rws,$\mathcal{O}_{\mathsf{h}}''$,$\mathcal{R}_{\mathsf{h}}''$,$\mathcal{S}_{\mathsf{h}}''$) $\rightarrow_{\mathsf{f}}^*$ (m(a:=v), [],$\mathcal{O}_{\mathsf{h}}'''$,$\mathcal{R}_{\mathsf{h}}'''$,$\mathcal{S}_{\mathsf{h}}'''$) **by** blast
with steps
have (m, sb_{h},$\mathcal{O}_{\mathsf{h}}$,$\mathcal{R}_{\mathsf{h}}$,$\mathcal{S}_{\mathsf{h}}$) $\rightarrow_{\mathsf{f}}^*$ (m(a:=v), [],$\mathcal{O}_{\mathsf{h}}'''$,$\mathcal{R}_{\mathsf{h}}'''$,$\mathcal{S}_{\mathsf{h}}'''$) **by** force
with sb Nil **show** ?thesis **by** fastforce
qed
qed

lemma bufferd-val-filter-is-$\text{Write}_{\mathsf{sb}}$-eq-ext:
buffered-val (filter is-$\text{Write}_{\mathsf{sb}}$ sb) a = buffered-val sb a
by (induct sb) (auto split: memref.splits)

lemma bufferd-val-filter-is-$\text{Write}_{\mathsf{sb}}$-eq:
buffered-val (filter is-$\text{Write}_{\mathsf{sb}}$ sb) = buffered-val sb
by (rule ext) (rule bufferd-val-filter-is-$\text{Write}_{\mathsf{sb}}$-eq-ext)

lemma outstanding-refs-is-volatile-$\text{Write}_{\mathsf{sb}}$-filter-writes:
outstanding-refs is-volatile-$\text{Write}_{\mathsf{sb}}$ (filter is-$\text{Write}_{\mathsf{sb}}$ xs) =
outstanding-refs is-volatile-$\text{Write}_{\mathsf{sb}}$ xs
by (induct xs) (auto simp add: is-volatile-$\text{Write}_{\mathsf{sb}}$-def split: memref.splits)

A.6 Simulation of Store Buffer Machine without History by Store Buffer Machine with History

theorem (**in** valid-program) concurrent-history-steps-simulates-store-buffer-step:
assumes step-sb: (ts,m,$\mathcal{S}$) $\Rightarrow_{\mathsf{sb}}$ (ts$'$,m$'$,$\mathcal{S}'$)
assumes sim: ts $\sim_{\mathsf{h}}$ ts_{h}
shows $\exists$$\text{ts}_{\mathsf{h}}'$ $\mathcal{S}_{\mathsf{h}}'$. ($\text{ts}_{\mathsf{h}}$,m,$\mathcal{S}_{\mathsf{h}}$) $\Rightarrow_{\mathsf{sbh}}^*$ (ts_{h}',m$'$,$\mathcal{S}_{\mathsf{h}}'$) $\wedge$ ts$'$ $\sim_{\mathsf{h}}$ ts_{h}'
proof –

interpret sbh-computation:
computation sbh-memop-step flush-step program-step
λp p′ is sb. sb @ [Prog$_{\mathsf{sb}}$ p p′ is] **.**
from step-sb
show ?thesis
proof (cases rule: concurrent-step-cases)
case (Memop i - p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ - - is′ j′ sb′ - $\mathcal{D}'$ $\mathcal{O}'$ $\mathcal{R}'$)
then obtain
ts′: ts′ = ts[i := (p, is′, j′, sb′, $\mathcal{D}'$, $\mathcal{O}'$,$\mathcal{R}'$)] **and**
i-bound: i < length ts **and**
ts-i: ts ! i = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) **and**
step-sb: (is, j, sb, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{sb}}$
(is′, j′, sb′, m′, $\mathcal{D}'$, $\mathcal{O}'$, $\mathcal{R}'$,$\mathcal{S}'$)
by auto

from sim **obtain**
lts-eq: length ts = length ts$_{\mathsf{h}}$ **and**
sim-loc: $\forall$ i < length ts. ($\exists$ $\mathcal{O}'$ $\mathcal{D}'$ $\mathcal{R}'$.
let (p,is, j, sb,$\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) = ts$_{\mathsf{h}}$!i in
ts!i=(p,is, j, filter is-Write$_{\mathsf{sb}}$ sb,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$) $\wedge$
(filter is-Write$_{\mathsf{sb}}$ sb = [] $\longrightarrow$ sb=[]))
by cases (auto)

from lts-eq i-bound **have** i-bound′: i < length ts$_{\mathsf{h}}$
by simp

from step-sb
show ?thesis
proof (cases)
case (SBReadBuffered a v volatile t)
then obtain
is: is = Read volatile a t#is′ **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=$\mathcal{R}$ **and**
$\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**
m′: m′=m **and**
j′: j′=j(t$\mapsto$v) **and**
sb′: sb′ = sb **and**
buf-val: buffered-val sb a = Some v
by auto

from sim-loc [rule-format, OF i-bound] ts-i is
obtain sb$_{\mathsf{h}}$ $\mathcal{O}_{\mathsf{h}}$ $\mathcal{R}_{\mathsf{h}}$ $\mathcal{D}_{\mathsf{h}}$ **where**
ts$_{\mathsf{h}}$-i: ts$_{\mathsf{h}}$!i = (p,Read volatile a t#is′,j,sb$_{\mathsf{h}}$,$\mathcal{D}_{\mathsf{h}}$,$\mathcal{O}_{\mathsf{h}}$,$\mathcal{R}_{\mathsf{h}}$) **and**
sb: sb = filter is-Write$_{\mathsf{sb}}$ sb$_{\mathsf{h}}$ **and**
sb-empty: filter is-Write$_{\mathsf{sb}}$ sb$_{\mathsf{h}}$ = [] $\longrightarrow$ sb$_{\mathsf{h}}$=[]
by (auto simp add: Let-def)

from buf-val

have buf-val′: buffered-val sb_h a = Some v
by (simp add: bufferd-val-filter-is-$Write_{sb}$-eq sb)

let ?ts_h-i′ = (p, is′, j(t ↦ v), sb_h @ [$Read_{sb}$ volatile a t v], $\mathcal{D}_h$, $\mathcal{O}_h$,$\mathcal{R}_h$)
let ?ts_h′ = ts_h[i := ?ts_h-i′]
from sbh-memop-step.SBHReadBuffered [OF buf-val′]
have (Read volatile a t # is′, j, sb_h, m,$\mathcal{D}_h$, $\mathcal{O}_h$, $\mathcal{R}_h$,$\mathcal{S}_h$) $\rightarrow_{sbh}$
(is′, j(t ↦ v), sb_h@ [$Read_{sb}$ volatile a t v], m, $\mathcal{D}_h$, $\mathcal{O}_h$, $\mathcal{R}_h$, $\mathcal{S}_h$).
from sbh-computation.Memop [OF i-bound′ ts_h-i this]
have step: (ts_h, m, $\mathcal{S}_h$) $\Rightarrow_{sbh}$ (?ts_h′, m, $\mathcal{S}_h$).

from sb **have** sb: sb = filter is-$Write_{sb}$ (sb_h @ [$Read_{sb}$ volatile a t v])
by simp

show ?thesis
proof (cases filter is-$Write_{sb}$ sb_h = [])
case False

have ts [i := (p,is′,j(t ↦ v),sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] $\sim_h$?ts_h′
apply (rule sim-history-config.intros)
using lts-eq
apply simp
using sim-loc i-bound i-bound′ sb sb-empty False
apply (auto simp add: Let-def nth-list-update)
done

with step **show** ?thesis
by (auto simp del: fun-upd-apply simp add: $\mathcal{S}'$ m′ ts′ $\mathcal{O}'$ j′ $\mathcal{D}'$ sb′ $\mathcal{R}'$)
next
case True
with sb-empty **have** empty: sb_h=[] **by** simp
from i-bound′ **have** ?ts_h′!i = ?ts_h-i′
by auto

from sbh-computation.StoreBuffer [OF - this, simplified empty, simplified, OF - flush-step.$Read_{sb}$, of m $\mathcal{S}_h$] i-bound′
have (?ts_h′, m, $\mathcal{S}_h$)
$\Rightarrow_{sbh}$ (ts_h[i := (p, is′, j(t ↦ v), [], $\mathcal{D}_h$, $\mathcal{O}_h$,$\mathcal{R}_h$)], m, $\mathcal{S}_h$)
by (simp add: empty list-update-overwrite)
with step **have** (ts_h, m, $\mathcal{S}_h$) $\Rightarrow_{sbh}^*$
(ts_h[i := (p, is′, j(t ↦ v), [], $\mathcal{D}_h$, $\mathcal{O}_h$,$\mathcal{R}_h$)], m,$\mathcal{S}_h$)
by force
moreover
have ts [i := (p,is′,j(t ↦ v),sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] $\sim_h$ ts_h[i := (p, is′, j(t ↦ v), [], $\mathcal{D}_h$, $\mathcal{O}_h$,$\mathcal{R}_h$)]
apply (rule sim-history-config.intros)
using lts-eq
apply simp
using sim-loc i-bound i-bound′ sb empty
apply (auto simp add: Let-def nth-list-update)

done
ultimately show ?thesis
by (auto simp del: fun-upd-apply simp add: $\mathcal{S}'$ m′ ts′ $\mathcal{O}'$ j′ $\mathcal{D}'$ sb′ $\mathcal{R}'$)
qed
next
case (SBReadUnbuffered a volatile t)
then obtain
is: is = Read volatile a t#is′ **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=$\mathcal{R}$ **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ **and**
$\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**
m′: m′=m **and**
j′: j′=j(t↦m a) **and**
sb′: sb′ = sb **and**
buf: buffered-val sb a = None
by auto

from sim-loc [rule-format, OF i-bound] ts-i is
obtain sb_h $\mathcal{O}_h$ $\mathcal{R}_h$ $\mathcal{D}_h$ **where**
ts_h-i: ts_h!i = (p,Read volatile a t#is′,j,sb_h,$\mathcal{D}_h$,$\mathcal{O}_h$,$\mathcal{R}_h$) **and**
sb: sb = filter is-$Write_{sb}$ sb_h **and**
sb-empty: filter is-$Write_{sb}$ sb_h = [] ⟶ sb_h=[]
by (auto simp add: Let-def)

from buf
have buf′: buffered-val sb_h a = None
by (simp add: bufferd-val-filter-is-$Write_{sb}$-eq sb)

let ?ts_h-i′ = (p, is′, j(t ↦ m a), sb_h @ [$Read_{sb}$ volatile a t (m a)], $\mathcal{D}_h$, $\mathcal{O}_h$,$\mathcal{R}_h$)
let ?ts_h′ = ts_h[i := ?ts_h-i′]

from sbh-memop-step.SBHReadUnbuffered [OF buf′]
have (Read volatile a t # is′,j, sb_h, m, $\mathcal{D}_h$, $\mathcal{O}_h$, $\mathcal{R}_h$,$\mathcal{S}_h$) $\rightarrow_{sbh}$
(is′, j(t ↦ (m a)), sb_h@ [$Read_{sb}$ volatile a t (m a)], m,$\mathcal{D}_h$, $\mathcal{O}_h$, $\mathcal{R}_h$,$\mathcal{S}_h$).
from sbh-computation.Memop [OF i-bound′ ts_h-i this]
have step: (ts_h, m, $\mathcal{S}_h$) $\Rightarrow_{sbh}$
(?ts_h′, m, $\mathcal{S}_h$).
moreover
from sb **have** sb: sb = filter is-$Write_{sb}$ (sb_h @ [$Read_{sb}$ volatile a t (m a)])
by simp

show ?thesis
proof (cases filter is-$Write_{sb}$ sb_h = [])
case False
have ts [i := (p,is′,j (t↦m a),sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] $\sim_h$?ts_h′
apply (rule sim-history-config.intros)
using lts-eq
apply simp
using sim-loc i-bound i-bound′ sb sb-empty False

apply (auto simp add: Let-def nth-list-update)
done

with step **show** ?thesis
by (auto simp del: fun-upd-apply simp add: $\mathcal{S}'$ m′ ts′ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{D}'$ j′ sb′)
next
case True
with sb-empty **have** empty: sb_h=[] **by** simp
from i-bound′ **have** ?ts_h′!i = ?ts_h-i′
by auto

from sbh-computation.StoreBuffer [OF - this, simplified empty, simplified, OF - flush-step.$Read_{sb}$, of m $\mathcal{S}_h$] i-bound′
have (?ts_h′, m, $\mathcal{S}_h$)
$\Rightarrow_{sbh}$ (ts_h[i := (p, is′, j(t ↦ (m a)), [], $\mathcal{D}_h$, $\mathcal{O}_h$,$\mathcal{R}_h$)], m, $\mathcal{S}_h$)
by (simp add: empty)
with step **have** (ts_h, m, $\mathcal{S}_h$) $\Rightarrow_{sbh}{}^*$
(ts_h[i := (p, is′, j(t ↦ m a), [], $\mathcal{D}_h$, $\mathcal{O}_h$,$\mathcal{R}_h$)], m, $\mathcal{S}_h$)
by force
moreover
have ts [i := (p,is′,j(t ↦ m a),sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] $\sim_h$ ts_h[i := (p, is′, j(t ↦ m a), [], $\mathcal{D}_h$, $\mathcal{O}_h$,$\mathcal{R}_h$)]
apply (rule sim-history-config.intros)
using lts-eq
apply simp
using sim-loc i-bound i-bound′ sb empty
apply (auto simp add: Let-def nth-list-update)
done
ultimately show ?thesis
by (auto simp del: fun-upd-apply simp add: $\mathcal{S}'$ m′ ts′ $\mathcal{O}'$ j′ $\mathcal{D}'$ sb′ $\mathcal{R}'$)
qed
next
case (SBWriteNonVolatile a D f A L R W)
then obtain
is: is = Write False a (D, f) A L R W#is′ **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=$\mathcal{R}$ **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ **and**
$\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**
m′: m′=m **and**
j′: j′=j **and**
sb′: sb′ = sb@[$Write_{sb}$ False a (D, f) (f j) A L R W]
by auto

from sim-loc [rule-format, OF i-bound] ts-i
obtain sb_h $\mathcal{O}_h$ $\mathcal{R}_h$ $\mathcal{D}_h$ **where**
ts_h-i: ts_h!i = (p,Write False a (D,f) A L R W#is′,j,sb_h,$\mathcal{D}_h$,$\mathcal{O}_h$,$\mathcal{R}_h$) **and**
sb: sb = filter is-$Write_{sb}$ sb_h
by (auto simp add: Let-def is)

from sbh-memop-step.SBHWriteNonVolatile
have (Write False a (D, f) A L R W# is$'$,j, sb$_h$, m, $\mathcal{D}_h$, $\mathcal{O}_h$, $\mathcal{R}_h$,$\mathcal{S}_h$) $\rightarrow_{sbh}$
(is$'$, j, sb$_h$ @ [Write$_{sb}$ False a (D, f) (f j) A L R W], m,$\mathcal{D}_h$, $\mathcal{O}_h$, $\mathcal{R}_h$,$\mathcal{S}_h$).
from sbh-computation.Memop [OF i-bound$'$ ts$_h$-i this]
have (ts$_h$, m, $\mathcal{S}_h$) $\Rightarrow_{sbh}$
(ts$_h$[i := (p, is$'$,j, sb$_h$ @ [Write$_{sb}$ False a (D, f) (f j) A L R W], $\mathcal{D}_h$, $\mathcal{O}_h$,$\mathcal{R}_h$)],
m, $\mathcal{S}_h$).
moreover
have ts [i := (p,is$'$,j,sb @ [Write$_{sb}$ False a (D,f) (f j) A L R W],$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] $\sim_h$
ts$_h$[i := (p,is$'$,j, sb$_h$ @ [Write$_{sb}$ False a (D,f) (f j) A L R W],$\mathcal{D}_h$, $\mathcal{O}_h$,$\mathcal{R}_h$)]
apply (rule sim-history-config.intros)
using lts-eq
apply simp
using sim-loc i-bound i-bound$'$ sb
apply (auto simp add: Let-def nth-list-update)
done

ultimately show ?thesis
by (auto simp add: $\mathcal{S}'$ m$'$ j$'$ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{D}'$ ts$'$ sb$'$)
next
case (SBWriteVolatile a D f A L R W)
then obtain
is: is = Write True a (D, f) A L R W#is$'$ **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=$\mathcal{R}$ **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ **and**
$\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**
m$'$: m$'$=m **and**
j$'$: j$'$=j **and**
sb$'$: sb$'$ = sb@[Write$_{sb}$ True a (D, f) (f j) A L R W]
by auto

from sim-loc [rule-format, OF i-bound] ts-i is
obtain sb$_h$ $\mathcal{O}_h$ $\mathcal{R}_h$ $\mathcal{D}_h$ **where**
ts$_h$-i: ts$_h$!i = (p,Write True a (D,f) A L R W#is$'$,j,sb$_h$,$\mathcal{D}_h$,$\mathcal{O}_h$,$\mathcal{R}_h$) **and**
sb: sb = filter is-Write$_{sb}$ sb$_h$
by (auto simp add: Let-def)

from sbh-computation.Memop [OF i-bound$'$ ts$_h$-i SBHWriteVolatile
]
have (ts$_h$, m, $\mathcal{S}_h$) $\Rightarrow_{sbh}$
(ts$_h$[i := (p, is$'$,j, sb$_h$ @ [Write$_{sb}$ True a (D, f) (f j) A L R W], True, $\mathcal{O}_h$,$\mathcal{R}_h$)],
m, $\mathcal{S}_h$).

moreover
have ts [i := (p,is$'$,j,sb @ [Write$_{sb}$ True a (D,f) (f j) A L R W],$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] $\sim_h$
ts$_h$[i := (p,is$'$, j, sb$_h$ @ [Write$_{sb}$ True a (D,f) (f j) A L R W],True, $\mathcal{O}_h$,$\mathcal{R}_h$)]
apply (rule sim-history-config.intros)
using lts-eq

apply simp
using sim-loc i-bound i-bound′ sb
apply (auto simp add: Let-def nth-list-update)
done

ultimately show ?thesis
by (auto simp add: ts′ $\mathcal{O}'$ j′ m′ sb′ $\mathcal{D}'$ $\mathcal{R}'$ $\mathcal{S}'$)
next
case SBFence
then obtain
is: is = Fence #is′ **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=$\mathcal{R}$ **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ **and**
$\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**
m′: m′=m **and**
j′: j′=j **and**
sb: sb = [] **and**
sb′: sb′ = []
by auto

from sim-loc [rule-format, OF i-bound] ts-i sb is
obtain sb_h $\mathcal{O}_h$ $\mathcal{R}_h$ $\mathcal{D}_h$ **where**
ts_h-i: ts_h!i = (p,Fence # is′,j,sb_h,$\mathcal{D}_h$,$\mathcal{O}_h$,$\mathcal{R}_h$) **and**
sb: [] = filter is-$Write_{sb}$ sb_h
by (auto simp add: Let-def)

from filter-is-$Write_{sb}$-empty [OF sb [symmetric]]
have $\forall$ r $\in$ set sb_h. is-$Read_{sb}$ r $\vee$ is-$Prog_{sb}$ r $\vee$ is-$Ghost_{sb}$ r.

from flush-reads-program [OF this] **obtain** $\mathcal{O}_h{}'$ $\mathcal{S}_h{}'$ $\mathcal{R}_h{}'$
where flsh: (m, sb_h,$\mathcal{O}_h$,$\mathcal{R}_h$,$\mathcal{S}_h$) $\rightarrow_f{}^*$ (m, [],$\mathcal{O}_h{}'$,$\mathcal{R}_h{}'$,$\mathcal{S}_h{}'$) **by** blast

let ?$ts_h{}'$ = ts_h[i := (p,Fence # is′, j, [], $\mathcal{D}_h$, $\mathcal{O}_h{}'$,$\mathcal{R}_h{}'$)]
from sbh-computation.store-buffer-steps [OF flsh i-bound′ ts_h-i]
have (ts_h, m, $\mathcal{S}_h$) $\Rightarrow_{sbh}{}^*$ (?$ts_h{}'$, m, $\mathcal{S}_h{}'$).

also

from i-bound′ **have** i-bound″: i < length ?$ts_h{}'$
by auto

from i-bound′ **have** $ts_h{}'$-i: ?$ts_h{}'$!i = (p,Fence#is′,j,[],$\mathcal{D}_h$,$\mathcal{O}_h{}'$,$\mathcal{R}_h{}'$)
by simp
from sbh-computation.Memop [OF i-bound″ $ts_h{}'$-i SBHFence] i-bound′
have (?$ts_h{}'$, m, $\mathcal{S}_h{}'$) $\Rightarrow_{sbh}$ (ts_h[i := (p, is′,j, [], False, $\mathcal{O}_h{}'$,Map.empty)], m,$\mathcal{S}_h{}'$)
by (simp)
finally
have (ts_h, m, $\mathcal{S}_h$) $\Rightarrow_{sbh}{}^*$ (ts_h[i := (p, is′, j, [],False, $\mathcal{O}_h{}'$,Map.empty)],m, $\mathcal{S}_h{}'$).

moreover

have ts [i := (p,is′,j,[],$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] $\sim_h$ ts_h[i := (p,is′, j, [],False, $\mathcal{O}_h$′,Map.empty)]
apply (rule sim-history-config.intros)
using lts-eq
apply simp
using sim-loc i-bound i-bound′ sb
apply (auto simp add: Let-def nth-list-update)
done

ultimately show ?thesis
by (auto simp add: ts′ $\mathcal{O}$′ j′ m′ sb′ $\mathcal{D}$′ $\mathcal{S}$′ $\mathcal{R}$′)

next
case (SBRMWReadOnly cond t a D f ret A L R W)
then obtain
is: is = RMW a t (D, f) cond ret A L R W#is′ **and**
$\mathcal{O}$′: $\mathcal{O}$′=$\mathcal{O}$ **and**
$\mathcal{R}$′: $\mathcal{R}$′=$\mathcal{R}$ **and**
$\mathcal{S}$′: $\mathcal{S}$′=$\mathcal{S}$ **and**
$\mathcal{D}$′: $\mathcal{D}$′=$\mathcal{D}$ **and**
m′: m′=m **and**
j′: j′=j(t ↦ m a) **and**
sb: sb=[] **and**
sb′: sb′ = [] **and**
cond: ¬ cond (j(t ↦ m a))
by auto

from sim-loc [rule-format, OF i-bound] ts-i sb is
obtain sb_h $\mathcal{O}_h$ $\mathcal{R}_h$ $\mathcal{D}_h$ **where**
ts_h-i: ts_h!i = (p,RMW a t (D, f) cond ret A L R W# is′,j,sb_h,$\mathcal{D}_h$,$\mathcal{O}_h$,$\mathcal{R}_h$) **and**
sb: [] = filter is-$Write_{sb}$ sb_h
by (auto simp add: Let-def)

from filter-is-$Write_{sb}$-empty [OF sb [symmetric]]
have ∀ r ∈ set sb_h. is-$Read_{sb}$ r ∨ is-$Prog_{sb}$ r ∨ is-$Ghost_{sb}$ r.

from flush-reads-program [OF this] **obtain** $\mathcal{O}_h$′ $\mathcal{S}_h$′ $\mathcal{R}_h$′
where flsh: (m, sb_h,$\mathcal{O}_h$,$\mathcal{R}_h$,$\mathcal{S}_h$) $\rightarrow_f^*$ (m, [],$\mathcal{O}_h$′,$\mathcal{R}_h$′,$\mathcal{S}_h$′) **by** blast

let ?ts_h′ = ts_h[i := (p,RMW a t (D, f) cond ret A L R W# is′,j, [], $\mathcal{D}_h$, $\mathcal{O}_h$′,$\mathcal{R}_h$′)]
from sbh-computation.store-buffer-steps [OF flsh i-bound′ ts_h-i]
have (ts_h, m, $\mathcal{S}_h$) $\Rightarrow_{sbh}^*$ (?ts_h′, m, $\mathcal{S}_h$′).

also

from i-bound′ **have** i-bound″: i < length ?ts_h′

by auto

from i-bound′ **have** ts_h′-i: ?ts_h′!i = (p,RMW a t (D, f) cond ret A L R W#is′,j,[],$\mathcal{D}_h$,$\mathcal{O}_h$′,$\mathcal{R}_h$′)
by simp

note step= SBHRMWReadOnly [**where** cond=cond **and** j=j **and** m=m, OF cond]
from sbh-computation.Memop [OF i-bound″ ts_h′-i step] i-bound′
have (?ts_h′, m, $\mathcal{S}_h$′) $\Rightarrow_{sbh}$ (ts_h[i := (p, is′,j(t↦m a), [], False, $\mathcal{O}_h$′,Map.empty)],m, $\mathcal{S}_h$′)
by (simp)
finally
have (ts_h, m, $\mathcal{S}_h$) $\Rightarrow_{sbh}^*$ (ts_h[i := (p, is′,j(t↦m a), [], False, $\mathcal{O}_h$′,Map.empty)],m, $\mathcal{S}_h$′).

moreover

have ts [i := (p,is′,j(t↦m a),[],$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] $\sim_h$ ts_h[i := (p,is′, j(t↦m a), [], False, $\mathcal{O}_h$′,Map.empty)]
apply (rule sim-history-config.intros)
using lts-eq
apply simp
using sim-loc i-bound i-bound′ sb
apply (auto simp add: Let-def nth-list-update)
done

ultimately show ?thesis
by (auto simp add: ts′ $\mathcal{O}$′ j′ m′ sb′ $\mathcal{D}$′ $\mathcal{S}$′ $\mathcal{R}$′)
next
case (SBRMWWrite cond t a D f ret A L R W)
then obtain
is: is = RMW a t (D, f) cond ret A L R W#is′ **and**
$\mathcal{O}$′: $\mathcal{O}$′=$\mathcal{O}$ **and**
$\mathcal{R}$′: $\mathcal{R}$′=$\mathcal{R}$ **and**
$\mathcal{S}$′: $\mathcal{S}$′=$\mathcal{S}$ **and**
$\mathcal{D}$′: $\mathcal{D}$′=$\mathcal{D}$ **and**
m′: m′=m(a := f (j(t ↦ (m a)))) **and**
j′: j′=j(t ↦ ret (m a) (f (j(t ↦ (m a))))) **and**
sb: sb=[] **and**
sb′: sb′ = [] **and**
cond: cond (j(t ↦ m a))
by auto

from sim-loc [rule-format, OF i-bound] ts-i sb is
obtain sb_h $\mathcal{O}_h$ $\mathcal{R}_h$ $\mathcal{D}_h$ acq_h **where**
ts_h-i: ts_h!i = (p,RMW a t (D, f) cond ret A L R W# is′,j,sb_h,$\mathcal{D}_h$,$\mathcal{O}_h$,$\mathcal{R}_h$) **and**
sb: [] = filter is-$Write_{sb}$ sb_h
by (auto simp add: Let-def)

from filter-is-$Write_{sb}$-empty [OF sb [symmetric]]

have $\forall$r $\in$ set sb_{h}. is-Read$_{\mathsf{sb}}$ r $\vee$ is-Prog$_{\mathsf{sb}}$ r $\vee$ is-Ghost$_{\mathsf{sb}}$ r**.**

from flush-reads-program [OF this] **obtain** $\mathcal{O}_{\mathsf{h}}{}'$ $\mathcal{S}_{\mathsf{h}}{}'$ $\mathcal{R}_{\mathsf{h}}{}'$
where flsh: (m, sb_{h},$\mathcal{O}_{\mathsf{h}}$,$\mathcal{R}_{\mathsf{h}}$,$\mathcal{S}_{\mathsf{h}}$) $\rightarrow_{\mathsf{f}}{}^*$ (m, [],$\mathcal{O}_{\mathsf{h}}{}'$,$\mathcal{R}_{\mathsf{h}}{}'$,$\mathcal{S}_{\mathsf{h}}{}'$) **by** blast

let ?$ts_{\mathsf{h}}{}'$ = ts_{h}[i := (p,RMW a t (D, f) cond ret A L R W# is′,j, [], $\mathcal{D}_{\mathsf{h}}$, $\mathcal{O}_{\mathsf{h}}{}'$,$\mathcal{R}_{\mathsf{h}}{}'$)]

from sbh-computation.store-buffer-steps [OF flsh i-bound′ ts_{h}-i]
have (ts_{h}, m, $\mathcal{S}_{\mathsf{h}}$) $\Rightarrow_{\mathsf{sbh}}{}^*$ (?$ts_{\mathsf{h}}{}'$, m, $\mathcal{S}_{\mathsf{h}}{}'$)**.**

also

from i-bound′ **have** i-bound″: i < length ?$ts_{\mathsf{h}}{}'$
by auto

from i-bound′ **have** $ts_{\mathsf{h}}{}'$-i: ?$ts_{\mathsf{h}}{}'$!i = (p,RMW a t (D, f) cond ret A L R W#is′,j,[],$\mathcal{D}_{\mathsf{h}}$,$\mathcal{O}_{\mathsf{h}}{}'$,$\mathcal{R}_{\mathsf{h}}{}'$)
by simp

note step= SBHRMWWrite [**where** cond=cond **and** j=j **and** m=m, OF cond]
from sbh-computation.Memop [OF i-bound″ $ts_{\mathsf{h}}{}'$-i step] i-bound′
have (?$ts_{\mathsf{h}}{}'$, m, $\mathcal{S}_{\mathsf{h}}{}'$) $\Rightarrow_{\mathsf{sbh}}$ (ts_{h}[i := (p, is′,
j(t $\mapsto$ ret (m a) (f (j(t $\mapsto$ (m a))))), [], False, $\mathcal{O}_{\mathsf{h}}{}'$ $\cup$ A $-$ R,Map.empty)],
m(a := f (j(t $\mapsto$ (m a)))),$\mathcal{S}_{\mathsf{h}}{}'$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
by (simp)
finally
have (ts_{h}, m, $\mathcal{S}_{\mathsf{h}}$) $\Rightarrow_{\mathsf{sbh}}{}^*$ (ts_{h}[i := (p, is′,
j(t $\mapsto$ ret (m a) (f (j(t $\mapsto$ (m a))))), [], False, $\mathcal{O}_{\mathsf{h}}{}'$ $\cup$ A $-$ R,Map.empty)],
m(a := f (j(t $\mapsto$ (m a)))),$\mathcal{S}_{\mathsf{h}}{}'$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)**.**

moreover

have ts [i := (p,is′,j(t $\mapsto$ ret (m a) (f (j(t $\mapsto$ (m a))))),[],$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] $\sim_{\mathsf{h}}$
ts_{h}[i := (p,is′,j(t $\mapsto$ ret (m a) (f (j(t $\mapsto$ (m a))))), [],False, $\mathcal{O}_{\mathsf{h}}{}'$ $\cup$ A $-$ R,Map.empty)]
apply (rule sim-history-config.intros)
using lts-eq
apply simp
using sim-loc i-bound i-bound′ sb
apply (auto simp add: Let-def nth-list-update)
done

ultimately show ?thesis
by (auto simp add: ts′ $\mathcal{O}'$ j′ m′ sb′ $\mathcal{D}'$ $\mathcal{S}'$ $\mathcal{R}'$)
next
case (SBGhost A L R W)
then obtain
is: is = Ghost A L R W#is′ **and**
$\mathcal{O}'$: $\mathcal{O}'$=$\mathcal{O}$ **and**
$\mathcal{R}'$: $\mathcal{R}'$=$\mathcal{R}$ **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ **and**

$\mathcal{D}'$: $\mathcal{D}'$=$\mathcal{D}$ **and**
m′: m′=m **and**
j′: j′=j **and**
sb′: sb′ = sb
by auto

from sim-loc [rule-format, OF i-bound] ts-i is
obtain sb_h $\mathcal{O}_h$ $\mathcal{R}_h$ $\mathcal{D}_h$ **where**
ts_h-i: ts_h!i = (p,Ghost A L R W# is′,j,sb_h,$\mathcal{D}_h$,$\mathcal{O}_h$,$\mathcal{R}_h$) **and**
sb: sb = filter is-$Write_{sb}$ sb_h **and**
sb-empty: filter is-$Write_{sb}$ sb_h = [] $\longrightarrow$ sb_h=[]
by (auto simp add: Let-def)

let ?ts_h-i′ = (p, is′, j, sb_h@[$Ghost_{sb}$ A L R W],$\mathcal{D}_h$, $\mathcal{O}_h$,$\mathcal{R}_h$)
let ?ts_h′ = ts_h[i := ?ts_h-i′]
note step= SBHGhost
from sbh-computation.Memop [OF i-bound′ ts_h-i step] i-bound′
have step: (ts_h, m, $\mathcal{S}_h$) $\Rightarrow_{sbh}$ (?ts_h′,m, $\mathcal{S}_h$)
by (simp)

from sb **have** sb: sb = filter is-$Write_{sb}$ (sb_h @ [$Ghost_{sb}$ A L R W])
by simp

show ?thesis
proof (cases filter is-$Write_{sb}$ sb_h = [])
case False

have ts [i := (p,is′,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] $\sim_h$?ts_h′
apply (rule sim-history-config.intros)
using lts-eq
apply simp
using sim-loc i-bound i-bound′ sb sb-empty False
apply (auto simp add: Let-def nth-list-update)
done

with step **show** ?thesis
by (auto simp del: fun-upd-apply simp add: $\mathcal{S}'$ m′ ts′ $\mathcal{O}'$ $\mathcal{D}'$ j′ sb′ $\mathcal{R}'$)
next
case True
with sb-empty **have** empty: sb_h=[] **by** simp
from i-bound′ **have** ?ts_h′!i = ?ts_h-i′
by auto
from sbh-computation.StoreBuffer [OF - this, simplified empty, simplified, OF - flush-step.Ghost, of m $\mathcal{S}_h$] i-bound′
have (?ts_h′, m, $\mathcal{S}_h$)
$\Rightarrow_{sbh}$ (ts_h[i := (p, is′, j, [], $\mathcal{D}_h$, $\mathcal{O}_h$ ∪ A − R,augment-rels (dom $\mathcal{S}_h$) R $\mathcal{R}_h$)], m, $\mathcal{S}_h$ $\oplus_W$ R $\ominus_A$ L)
by (simp add: empty)
with step **have** (ts_h, m, $\mathcal{S}_h$) $\Rightarrow_{sbh}^*$

(ts$_h$[i := (p, is′, j, [], $\mathcal{D}_h$, $\mathcal{O}_h$ ∪ A − R,augment-rels (dom $\mathcal{S}_h$) R $\mathcal{R}_h$)], m, $\mathcal{S}_h$
$\oplus_W$ R $\ominus_A$ L)
by force
moreover
have ts [i := (p,is′,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)] $\sim_h$
ts$_h$[i := (p, is′, j, [], $\mathcal{D}_h$, $\mathcal{O}_h$ ∪ A − R,augment-rels (dom $\mathcal{S}_h$) R $\mathcal{R}_h$)]
apply (rule sim-history-config.intros)
using lts-eq
apply simp
using sim-loc i-bound i-bound′ sb empty
apply (auto simp add: Let-def nth-list-update)
done
ultimately show ?thesis
by (auto simp del: fun-upd-apply simp add: $\mathcal{S}'$ m′ ts′ $\mathcal{O}'$ j′ $\mathcal{D}'$ sb′ $\mathcal{R}'$)
qed
qed
next
case (Program i - p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ p′ is′)
then obtain
ts′: ts′ = ts[i := (p′, is@is′,j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)] **and**
i-bound: i < length ts **and**
ts-i: ts ! i = (p, is, j,sb,$\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) **and**
prog-step: j⊢ p $\rightarrow_p$ (p′, is′) **and**
$\mathcal{S}'$: $\mathcal{S}'$=$\mathcal{S}$ **and**
m′: m′=m
by auto

from sim **obtain**
lts-eq: length ts = length ts$_h$ **and**
sim-loc: ∀ i < length ts. (∃ $\mathcal{O}'$ $\mathcal{D}'$ $\mathcal{R}'$.
let (p,is,j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) = ts$_h$!i in ts!i=(p,is, j, filter is-Write$_{sb}$ sb,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$) ∧
(filter is-Write$_{sb}$ sb = [] ⟶ sb = []))
by cases auto

from sim-loc [rule-format, OF i-bound] ts-i
obtain sb$_h$ $\mathcal{O}_h$ $\mathcal{R}_h$ $\mathcal{D}_h$ acq$_h$ **where**
ts$_h$-i: ts$_h$!i = (p,is,j,sb$_h$,$\mathcal{D}_h$,$\mathcal{O}_h$,$\mathcal{R}_h$) **and**
sb: sb = filter is-Write$_{sb}$ sb$_h$ **and**
sb-empty: filter is-Write$_{sb}$ sb$_h$ = [] ⟶ sb$_h$=[]
by (auto simp add: Let-def)

from lts-eq i-bound **have** i-bound′: i < length ts$_h$
by simp

let ?ts$_h$-i′ = (p′, is @ is′,j, sb$_h$ @ [Prog$_{sb}$ p p′ is′], $\mathcal{D}_h$, $\mathcal{O}_h$,$\mathcal{R}_h$)
let ?ts$_h$′ = ts$_h$[i := ?ts$_h$-i′]
from sbh-computation.Program [OF i-bound′ ts$_h$-i prog-step]
have step: (ts$_h$, m, $\mathcal{S}_h$) $\Rightarrow_{sbh}$ (?ts$_h$′,m, $\mathcal{S}_h$).

show ?thesis

proof (cases filter is-Write$_{\mathsf{sb}}$ sb$_{\mathsf{h}}$ = [])
case False
have ts[i := (p′, is@is′, j, sb,$\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)] $\sim_{\mathsf{h}}$?ts$_{\mathsf{h}}$′
apply (rule sim-history-config.intros)
using lts-eq
apply simp
using sim-loc i-bound i-bound′ sb False sb-empty
apply (auto simp add: Let-def nth-list-update)
done

with step **show** ?thesis
by (auto simp add: ts′ $\mathcal{S}$′ m′)
next
case True
with sb-empty **have** empty: sb$_{\mathsf{h}}$=[] **by** simp
from i-bound′ **have** ?ts$_{\mathsf{h}}$′!i = ?ts$_{\mathsf{h}}$-i′
by auto

from sbh-computation.StoreBuffer [OF - this, simplified empty, simplified, OF - flush-step.Prog$_{\mathsf{sb}}$, of m $\mathcal{S}_{\mathsf{h}}$] i-bound′
have (?ts$_{\mathsf{h}}$′, m, $\mathcal{S}_{\mathsf{h}}$)
$\Rightarrow_{\mathsf{sbh}}$ (ts$_{\mathsf{h}}$[i := (p′, is@is′, j, [], $\mathcal{D}_{\mathsf{h}}$, $\mathcal{O}_{\mathsf{h}}$,$\mathcal{R}_{\mathsf{h}}$)], m, $\mathcal{S}_{\mathsf{h}}$)
by (simp add: empty)
with step **have** (ts$_{\mathsf{h}}$, m, $\mathcal{S}_{\mathsf{h}}$) $\Rightarrow_{\mathsf{sbh}}^*$
(ts$_{\mathsf{h}}$[i := (p′, is@is′, j, [], $\mathcal{D}_{\mathsf{h}}$, $\mathcal{O}_{\mathsf{h}}$,$\mathcal{R}_{\mathsf{h}}$)], m, $\mathcal{S}_{\mathsf{h}}$)
by force
moreover
have ts[i := (p′, is@is′, j, sb,$\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)] $\sim_{\mathsf{h}}$ ts$_{\mathsf{h}}$[i := (p′, is@is′, j, [], $\mathcal{D}_{\mathsf{h}}$, $\mathcal{O}_{\mathsf{h}}$,$\mathcal{R}_{\mathsf{h}}$)]
apply (rule sim-history-config.intros)
using lts-eq
apply simp
using sim-loc i-bound i-bound′ sb empty
apply (auto simp add: Let-def nth-list-update)
done
ultimately show ?thesis
by (auto simp del: fun-upd-apply simp add: $\mathcal{S}$′ m′ ts′)
qed
next
case (StoreBuffer i - p is j sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$ - - - sb′ $\mathcal{O}$′ $\mathcal{R}$′)
then obtain
ts′: ts′ = ts[i := (p, is,j, sb′, $\mathcal{D}$, $\mathcal{O}$′,$\mathcal{R}$′)] **and**
i-bound: i < length ts **and**
ts-i: ts ! i = (p, is,j,sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) **and**
sb-step: (m,sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{w}}$ (m′, sb′,$\mathcal{O}$′,$\mathcal{R}$′,$\mathcal{S}$′)
by auto

from sim **obtain**
lts-eq: length ts = length ts$_{\mathsf{h}}$ **and**
sim-loc: $\forall$ i < length ts. ($\exists$ $\mathcal{O}$′ $\mathcal{D}$′ $\mathcal{R}$′.
let (p,is, j, sb,$\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) = ts$_{\mathsf{h}}$!i in ts!i=(p,is, j, filter is-Write$_{\mathsf{sb}}$ sb,$\mathcal{D}$′,$\mathcal{O}$′,$\mathcal{R}$′) $\wedge$

(filter is-Write$_{sb}$ sb = [] $\longrightarrow$ sb=[]))
by cases auto

from sim-loc [rule-format, OF i-bound] ts-i
obtain sb$_h$ $\mathcal{O}_h$ $\mathcal{R}_h$ $\mathcal{D}_h$ acq$_h$ **where**
ts$_h$-i: ts$_h$!i = (p,is,j,sb$_h$,$\mathcal{D}_h$,$\mathcal{O}_h$,$\mathcal{R}_h$) **and**
sb: sb = filter is-Write$_{sb}$ sb$_h$ **and**
sb-empty: filter is-Write$_{sb}$ sb$_h$ = [] $\longrightarrow$ sb$_h$=[]
by (auto simp add: Let-def)

from lts-eq i-bound **have** i-bound′: i < length ts$_h$
by simp

from flush-simulates-filter-writes [OF sb-step sb, of $\mathcal{O}_h$ $\mathcal{R}_h$ $\mathcal{S}_h$]
obtain sb$_h$′ $\mathcal{O}_h$′ $\mathcal{R}_h$′ $\mathcal{S}_h$′
where flush′: (m, sb$_h$,$\mathcal{O}_h$,$\mathcal{R}_h$,$\mathcal{S}_h$) $\rightarrow_f^*$ (m′, sb$_h$′,$\mathcal{O}_h$′,$\mathcal{R}_h$′,$\mathcal{S}_h$′) **and**
sb′: sb′ = filter is-Write$_{sb}$ sb$_h$′ **and**
sb′-empty: filter is-Write$_{sb}$ sb$_h$′ = [] $\longrightarrow$ sb$_h$′=[]
by auto

from sb-step **obtain** volatile a sop v A L R W **where** sb=Write$_{sb}$ volatile a sop v A L R W#sb′
by cases force
from sbh-computation.store-buffer-steps [OF flush′ i-bound′ ts$_h$-i]
have (ts$_h$, m, $\mathcal{S}_h$) $\Rightarrow_{sbh}^*$ (ts$_h$[i := (p, is, j, sb$_h$′,$\mathcal{D}_h$, $\mathcal{O}_h$′,$\mathcal{R}_h$′)], m′, $\mathcal{S}_h$′).

moreover
have ts[i := (p, is, j, sb′,$\mathcal{D}$, $\mathcal{O}$′,$\mathcal{R}$′)] $\sim_h$
ts$_h$[i := (p, is, j, sb$_h$′,$\mathcal{D}_h$, $\mathcal{O}_h$′,$\mathcal{R}_h$′)]
apply (rule sim-history-config.intros)
using lts-eq
apply simp
using sim-loc i-bound i-bound′ sb sb′ sb′-empty
apply (auto simp add: Let-def nth-list-update)
done

ultimately show ?thesis
by (auto simp add: ts′)
qed
qed

theorem (**in** valid-program) concurrent-history-steps-simulates-store-buffer-steps:
assumes step-sb: (ts,m,$\mathcal{S}$) $\Rightarrow_{sb}^*$ (ts′,m′,$\mathcal{S}$′)
shows $\bigwedge$ts$_h$ $\mathcal{S}_h$. ts $\sim_h$ ts$_h$ $\Longrightarrow$ $\exists$ts$_h$′ $\mathcal{S}_h$′. (ts$_h$,m,$\mathcal{S}_h$) $\Rightarrow_{sbh}^*$ (ts$_h$′,m′,$\mathcal{S}_h$′) $\wedge$ ts′ $\sim_h$ ts$_h$′
using step-sb
proof (induct rule: converse-rtranclp-induct-sbh-steps)
case refl **thus** ?case **by** auto
next

case (step ts m $\mathcal{S}$ ts$''$ m$''$ $\mathcal{S}''$)
have first: (ts,m,$\mathcal{S}$) $\Rightarrow_{\mathsf{sb}}$ (ts$''$,m$''$,$\mathcal{S}''$) **by** fact
have sim: ts $\sim_{\mathsf{h}}$ ts$_{\mathsf{h}}$ **by** fact
from concurrent-history-steps-simulates-store-buffer-step [OF first sim, of $\mathcal{S}_{\mathsf{h}}$]
obtain ts$_{\mathsf{h}}''$ $\mathcal{S}_{\mathsf{h}}''$ **where**
exec: (ts$_{\mathsf{h}}$,m,$\mathcal{S}_{\mathsf{h}}$) $\Rightarrow_{\mathsf{sbh}}{}^*$ (ts$_{\mathsf{h}}''$,m$''$,$\mathcal{S}_{\mathsf{h}}''$) **and** sim: ts$''$ $\sim_{\mathsf{h}}$ ts$_{\mathsf{h}}''$
by auto
from step.hyps (3) [OF sim, of $\mathcal{S}_{\mathsf{h}}''$]
obtain ts$_{\mathsf{h}}'$ $\mathcal{S}_{\mathsf{h}}'$ **where** exec-rest: (ts$_{\mathsf{h}}''$,m$''$,$\mathcal{S}_{\mathsf{h}}''$) $\Rightarrow_{\mathsf{sbh}}{}^*$ (ts$_{\mathsf{h}}'$,m$'$,$\mathcal{S}_{\mathsf{h}}'$) **and** sim$'$: ts$'$ $\sim_{\mathsf{h}}$ ts$_{\mathsf{h}}'$
by auto
note exec **also note** exec-rest
finally show ?case
using sim$'$ **by** blast
qed

theorem (**in** xvalid-program-progress) concurrent-direct-execution-simulates-store-buffer-execution:
assumes exec-sb: (ts$_{\mathsf{sb}}$,m$_{\mathsf{sb}}$,x) $\Rightarrow_{\mathsf{sb}}{}^*$ (ts$_{\mathsf{sb}}'$,m$_{\mathsf{sb}}'$,x$'$)
assumes init: initial$_{\mathsf{sb}}$ ts$_{\mathsf{sb}}$ $\mathcal{S}_{\mathsf{sb}}$
assumes valid: valid ts$_{\mathsf{sb}}$
assumes sim: (ts$_{\mathsf{sb}}$,m$_{\mathsf{sb}}$,$\mathcal{S}_{\mathsf{sb}}$) $\sim$ (ts,m,$\mathcal{S}$)
assumes safe: safe-reach-direct safe-free-flowing (ts,m,$\mathcal{S}$)
shows $\exists$ts$_{\mathsf{h}}'$ $\mathcal{S}_{\mathsf{h}}'$ ts$'$ m$'$ $\mathcal{S}'$.
(ts$_{\mathsf{sb}}$,m$_{\mathsf{sb}}$,$\mathcal{S}_{\mathsf{sb}}$) $\Rightarrow_{\mathsf{sbh}}{}^*$ (ts$_{\mathsf{h}}'$,m$_{\mathsf{sb}}'$,$\mathcal{S}_{\mathsf{h}}'$) $\wedge$
ts$_{\mathsf{sb}}'$ $\sim_{\mathsf{h}}$ ts$_{\mathsf{h}}'$ $\wedge$
(ts,m,$\mathcal{S}$) $\Rightarrow_{\mathsf{d}}{}^*$ (ts$'$,m$'$,$\mathcal{S}'$) $\wedge$
(ts$_{\mathsf{h}}'$,m$_{\mathsf{sb}}'$,$\mathcal{S}_{\mathsf{h}}'$) $\sim$ (ts$'$,m$'$,$\mathcal{S}'$)
proof −
from init **interpret** ini: initial$_{\mathsf{sb}}$ ts$_{\mathsf{sb}}$ $\mathcal{S}_{\mathsf{sb}}$ **.**
from concurrent-history-steps-simulates-store-buffer-steps [OF exec-sb ini.history-refl, of $\mathcal{S}_{\mathsf{sb}}$]
obtain ts$_{\mathsf{h}}'$ $\mathcal{S}_{\mathsf{h}}'$ **where**
sbh: (ts$_{\mathsf{sb}}$,m$_{\mathsf{sb}}$,$\mathcal{S}_{\mathsf{sb}}$) $\Rightarrow_{\mathsf{sbh}}{}^*$ (ts$_{\mathsf{h}}'$,m$_{\mathsf{sb}}'$,$\mathcal{S}_{\mathsf{h}}'$) **and**
sim-sbh: ts$_{\mathsf{sb}}'$ $\sim_{\mathsf{h}}$ ts$_{\mathsf{h}}'$
by auto
from concurrent-direct-execution-simulates-store-buffer-history-execution [OF sbh init valid sim safe]
obtain ts$'$ m$'$ $\mathcal{S}'$ **where**
d: (ts,m,$\mathcal{S}$) $\Rightarrow_{\mathsf{d}}{}^*$ (ts$'$,m$'$,$\mathcal{S}'$) **and**
d-sim: (ts$_{\mathsf{h}}'$,m$_{\mathsf{sb}}'$,$\mathcal{S}_{\mathsf{h}}'$) $\sim$ (ts$'$,m$'$,$\mathcal{S}'$)
by auto
with sbh sim-sbh **show** ?thesis **by** blast
qed

inductive sim-direct-config::
($'$p,$'$p store-buffer,$'$dirty,$'$owns,$'$rels) thread-config list $\Rightarrow$ ($'$p,unit,bool,$'$owns$'$,$'$rels$'$) thread-config list $\Rightarrow$ bool
(‹- $\sim_{\mathsf{d}}$ - › [60,60] 100)

```
where
  ⟦length ts = length tsd;
    ∀i < length ts.
        (∃𝒪' 𝒟' ℛ'.
          let (p,is, j,sb,𝒟, 𝒪,ℛ) = tsd!i in
               ts!i=(p,is, j, [] ,𝒟',𝒪',ℛ'))
  ⟧
    ⟹
     ts ∼d tsd

lemma empty-sb-sims:
assumes empty:
  ∀i p is xs sb 𝒟 𝒪 ℛ. i < length tssb ⟶ tssb!i=(p,is,xs,sb,𝒟,𝒪,ℛ)⟶ sb=[]
assumes sim-h: tssb ∼h tsh
assumes sim-d: (tsh,mh,𝒮h) ∼ (ts,m,𝒮)
shows tssb ∼d ts ∧ mh=m ∧ length tssb = length ts
proof−
  from sim-h empty
  have empty':
  ∀i p is xs sb 𝒟 𝒪 ℛ. i < length tsh ⟶ tsh!i=(p,is,xs,sb,𝒟,𝒪,ℛ)⟶ sb=[]
    apply (cases)
    apply clarsimp
    subgoal for i
    apply (drule-tac x=i in spec)
    apply (auto simp add: Let-def)
    done
    done
  from sim-h sim-config-emptyE [OF empty' sim-d]
  show ?thesis
    apply cases
    apply clarsimp
    apply (rule sim-direct-config.intros)
    apply  clarsimp
    apply clarsimp
    using empty'
    subgoal for i
    apply (drule-tac x=i in spec)
    apply (drule-tac x=i in spec)
    apply (drule-tac x=i in spec)
    apply (auto simp add: Let-def)
    done
    done
qed

lemma empty-d-sims:
assumes sim: tssb ∼d ts
shows ∃tsh. tssb ∼h tsh ∧ (tsh,m,𝒮) ∼ (ts,m,𝒮)
proof −
  from sim obtain
    leq: length tssb = length ts and
```

sim: $\forall$ i < length ts_{sb}.
    ($\exists \mathcal{O}' \mathcal{D}' \mathcal{R}'$.
        let (p,is, j,sb,$\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) = ts!i in
            ts_{sb}!i=(p,is, j, [] ,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$))
  by cases auto
define ts_h **where** $ts_h \equiv$ map (λ(p,is, j,sb,$\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$). (p,is, j,[]::'a memref list,$\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)) ts
have $ts_{sb} \sim_h ts_h$
  apply (rule sim-history-config.intros)
  using leq sim
  apply (auto simp add: ts_h-def Let-def leq)
  done
moreover
have empty:
$\forall$ i p is xs sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$. i < length $ts_h \longrightarrow ts_h$!i=(p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)$\longrightarrow$ sb=[]
  apply (clarsimp simp add: ts_h-def Let-def)
  subgoal for i
  apply (case-tac ts!i)
  apply auto
  done
  done

have (ts_h,m,$\mathcal{S}$) $\sim$ (ts,m,$\mathcal{S}$)
  apply (rule sim-config-emptyI [OF empty])
  apply (clarsimp simp add: ts_h-def)
  apply (clarsimp simp add: ts_h-def Let-def)
  subgoal for i
  apply (case-tac ts!i)
  apply auto
  done
  done
ultimately show ?thesis **by** blast
qed

theorem (**in** xvalid-program-progress) concurrent-direct-execution-simulates-store-buffer-execution-empty:
assumes exec-sb: (ts_{sb},m_{sb},x) $\Rightarrow_{sb}^*$ (ts_{sb}',m_{sb}',x')
assumes init: $initial_{sb}$ ts_{sb} $\mathcal{S}_{sb}$
assumes valid: valid ts_{sb}
assumes empty:
  $\forall$ i p is xs sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$. i < length $ts_{sb}' \longrightarrow ts_{sb}'$!i=(p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)$\longrightarrow$ sb=[]
assumes sim: (ts_{sb},m_{sb},$\mathcal{S}_{sb}$) $\sim$ (ts,m,$\mathcal{S}$)
assumes safe: safe-reach-direct safe-free-flowing (ts,m,$\mathcal{S}$)
shows $\exists$ ts' $\mathcal{S}'$.
    (ts,m,$\mathcal{S}$) $\Rightarrow_d^*$ (ts',m_{sb}',$\mathcal{S}'$) $\wedge$ $ts_{sb}' \sim_d$ ts'
proof −
from concurrent-direct-execution-simulates-store-buffer-execution [OF exec-sb init valid sim safe]
obtain ts_h' $\mathcal{S}_h'$ ts' m' $\mathcal{S}'$ **where**
  (ts_{sb},m_{sb},$\mathcal{S}_{sb}$) $\Rightarrow_{sbh}^*$ (ts_h',m_{sb}',$\mathcal{S}_h'$) **and**
  sim-h: $ts_{sb}' \sim_h ts_h'$ **and**

exec: $(ts,m,\mathcal{S}) \Rightarrow_{\mathsf{d}}{}^{*} (ts',m',\mathcal{S}')$ **and**
sim: $(ts_{\mathsf{h}}',m_{\mathsf{sb}}',\mathcal{S}_{\mathsf{h}}') \sim (ts',m',\mathcal{S}')$
by auto
from empty-sb-sims [OF empty sim-h sim]
obtain $ts_{\mathsf{sb}}' \sim_{\mathsf{d}} ts'$ $m_{\mathsf{sb}}' = m'$ length ts_{sb}' = length ts'
by auto
thus ?thesis
using exec
by blast
qed

locale $\text{initial}_{\mathsf{d}}$ = simple-ownership-distinct + read-only-unowned + unowned-shared +
fixes valid
assumes empty-is: $[\![$i < length ts; ts!i=$(p,is,xs,sb,\mathcal{D},\mathcal{O},\mathcal{R})]\!] \Longrightarrow$ is=[]
assumes empty-rels: $[\![$i < length ts; ts!i=$(p,is,xs,sb,\mathcal{D},\mathcal{O},\mathcal{R})]\!] \Longrightarrow \mathcal{R}$=Map.empty
assumes valid-init: valid (map $(\lambda(p,is, j,sb,\mathcal{D}, \mathcal{O},\mathcal{R}). (p,is, j,[],\mathcal{D}, \mathcal{O},\mathcal{R}))$ ts)

locale empty-store-buffers =
fixes ts::('p,'p store-buffer,bool,owns,rels) thread-config list
assumes empty-sb: $[\![$i < length ts; ts!i=$(p,is,xs,sb,\mathcal{D},\mathcal{O},\mathcal{R})]\!] \Longrightarrow$ sb=[]

lemma initial-d-sb:
assumes init: $\text{initial}_{\mathsf{d}}$ ts $\mathcal{S}$ valid
shows $\text{initial}_{\mathsf{sb}}$ (map $(\lambda(p,is, j,sb,\mathcal{D}, \mathcal{O},\mathcal{R}). (p,is, j,[],\mathcal{D}, \mathcal{O},\mathcal{R}))$ ts) $\mathcal{S}$
(**is** $\text{initial}_{\mathsf{sb}}$?map $\mathcal{S}$)
proof −
from init **interpret** ini: $\text{initial}_{\mathsf{d}}$ ts $\mathcal{S}$ **.**
show ?thesis
proof (intro-locales)
show simple-ownership-distinct ?map
apply (clarsimp simp add: simple-ownership-distinct-def)
subgoal for i j
apply (case-tac ts!i)
apply (case-tac ts!j)
apply (cut-tac i=i **and** j=j **in** ini.simple-ownership-distinct)
apply clarsimp
apply clarsimp
apply clarsimp
apply assumption
apply assumption
apply auto
done
done
next
show read-only-unowned $\mathcal{S}$?map
apply (clarsimp simp add: read-only-unowned-def)
subgoal for i
apply (case-tac ts!i)
apply (cut-tac i=i **in** ini.read-only-unowned)
apply clarsimp

```
    apply  assumption
    apply auto
    done
    done
  next
    show unowned-shared 𝒮 ?map
    apply (clarsimp simp add: unowned-shared-def′)
    apply (rule ini.unowned-shared′)
    apply clarsimp
    subgoal for a i
    apply (case-tac ts!i)
    apply auto
    done
    done
  next
    show initial_sb-axioms ?map
    apply (unfold-locales)
            subgoal for i
            apply (case-tac ts!i)
            apply simp
            done
           subgoal for i
           apply  (case-tac ts!i)
           apply  clarsimp
           apply  (rule-tac i=i in ini.empty-is)
           apply   clarsimp
           apply  fastforce
           done
    subgoal for i
    apply (case-tac ts!i)
    apply clarsimp
    apply (rule-tac i=i in ini.empty-rels)
    apply  clarsimp
    apply fastforce
    done
    done
  qed
qed
```

theorem (**in** xvalid-program-progress) store-buffer-execution-result-sequential-consistent:
assumes exec-sb: $(ts_{sb},m,x) \Rightarrow_{sb}^{*} (ts_{sb}',m',x')$
assumes empty′: empty-store-buffers ts_{sb}'
assumes sim: $ts_{sb} \sim_{d} ts$
assumes init: $initial_{d}$ ts $\mathcal{S}$ valid
assumes safe: safe-reach-direct safe-free-flowing $(ts,m,\mathcal{S})$
shows $\exists ts'\ \mathcal{S}'.$
$\quad (ts,m,\mathcal{S}) \Rightarrow_{d}^{*} (ts',m',\mathcal{S}') \wedge ts_{sb}' \sim_{d} ts'$
proof –
  from empty′
  have empty′:

$\forall$i p is xs sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$. i < length ts_{sb}' $\longrightarrow$ ts_{sb}'!i=(p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)$\longrightarrow$ sb=[]
  by (auto simp add: empty-store-buffers-def)

define ts_h **where** $ts_h \equiv$ map (λ(p,is, j,sb,$\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$). (p,is, j,[]::'a memref list,$\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)) ts
from initial-d-sb [OF init]
have init-h:initial_{sb} ts_h $\mathcal{S}$
  by (simp add: ts_h-def)
from initial_d.valid-init [OF init]
have valid-h: valid ts_h
  by (simp add: ts_h-def)
from sim **obtain**
  leq: length ts_{sb} = length ts **and**
  sim: $\forall$i < length ts_{sb}.
    ($\exists\mathcal{O}'$ $\mathcal{D}'$ $\mathcal{R}'$.
      let (p,is, j,sb,$\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$) = ts!i in
        ts_{sb}!i=(p,is, j, [] ,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$))
  by cases auto
have sim-h: ts_{sb} $\sim_h$ ts_h
  apply (rule sim-history-config.intros)
  using leq sim
  apply (auto simp add: ts_h-def Let-def leq)
  done

from concurrent-history-steps-simulates-store-buffer-steps [OF exec-sb sim-h, of $\mathcal{S}$]
obtain ts_h' $\mathcal{S}_h'$ **where** steps-h: (ts_h,m,$\mathcal{S}$) $\Rightarrow_{sbh}^*$ (ts_h',m',$\mathcal{S}_h'$) **and**
   sim-h': ts_{sb}' $\sim_h$ ts_h'
  by auto

moreover
have empty:
$\forall$i p is xs sb $\mathcal{D}$ $\mathcal{O}$ $\mathcal{R}$. i < length ts_h $\longrightarrow$ ts_h!i=(p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)$\longrightarrow$ sb=[]
  apply (clarsimp simp add: ts_h-def Let-def)
  subgoal for i
  apply (case-tac ts!i)
  apply auto
  done
  done

have sim': (ts_h,m,$\mathcal{S}$) $\sim$ (ts,m,$\mathcal{S}$)
  apply (rule sim-config-emptyI [OF empty])
  apply (clarsimp simp add: ts_h-def)
  apply (clarsimp simp add: ts_h-def Let-def)
  subgoal for i
  apply (case-tac ts!i)
  apply auto
  done
done

 from concurrent-direct-execution-simulates-store-buffer-history-execution [OF steps-h init-h valid-h sim' safe]

obtain ts′ m″ $\mathcal{S}''$ **where** steps: (ts, m, $\mathcal{S}$) $\Rightarrow_{\mathsf{d}}{}^*$ (ts′, m″, $\mathcal{S}''$)
and sim′: (ts$_{\mathsf{h}}$′, m′, $\mathcal{S}_{\mathsf{h}}$′) ∼ (ts′, m″, $\mathcal{S}''$)
by blast
from empty-sb-sims [OF empty′ sim-h′ sim′] steps
show ?thesis
by auto
qed

locale initial$_{\mathsf{v}}$ = simple-ownership-distinct + read-only-unowned + unowned-shared +
fixes valid
assumes empty-is: ⟦i < length ts; ts!i=(p,is,xs,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)⟧ ⟹ is=[]
assumes valid-init: valid (map (λ(p,is, j,sb,$\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$). (p,is, j,[],$\mathcal{D}$, $\mathcal{O}$,Map.empty)) ts)

theorem (**in** xvalid-program-progress) store-buffer-execution-result-sequential-consistent′:
assumes exec-sb: (ts$_{\mathsf{sb}}$,m,x) $\Rightarrow_{\mathsf{sb}}{}^*$ (ts$_{\mathsf{sb}}$′,m′,x′)
assumes empty′: empty-store-buffers ts$_{\mathsf{sb}}$′
assumes sim: ts$_{\mathsf{sb}}$ $\sim_{\mathsf{d}}$ ts
assumes init: initial$_{\mathsf{v}}$ ts $\mathcal{S}$ valid
assumes safe: safe-reach-virtual safe-free-flowing (ts,m,$\mathcal{S}$)
shows ∃ ts′ $\mathcal{S}$′.
(ts,m,$\mathcal{S}$) $\Rightarrow_{\mathsf{v}}{}^*$ (ts′,m′,$\mathcal{S}$′) ∧ ts$_{\mathsf{sb}}$′ $\sim_{\mathsf{d}}$ ts′
proof −
define ts$_{\mathsf{d}}$ **where** ts$_{\mathsf{d}}$ == (map (λ(p,is, j,sb,$\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$′). (p,is, j,sb,$\mathcal{D}$, $\mathcal{O}$,Map.empty::rels)) ts)
have rem-ts: remove-rels ts$_{\mathsf{d}}$ = ts
apply (rule nth-equalityI)
apply (simp add: ts$_{\mathsf{d}}$-def remove-rels-def)
apply (clarsimp simp add: ts$_{\mathsf{d}}$-def remove-rels-def)
subgoal for i
apply (case-tac ts!i)
apply clarsimp
done
done
from sim
have sim′: ts$_{\mathsf{sb}}$ $\sim_{\mathsf{d}}$ ts$_{\mathsf{d}}$
apply cases
apply (rule sim-direct-config.intros)
apply (auto simp add: ts$_{\mathsf{d}}$-def)
done

have init′: initial$_{\mathsf{d}}$ ts$_{\mathsf{d}}$ $\mathcal{S}$ valid
proof (intro-locales)
from init **have** simple-ownership-distinct ts
by (simp add: initial$_{\mathsf{v}}$-def)
then

```
  show simple-ownership-distinct tsd
    apply (clarsimp simp add: tsd-def simple-ownership-distinct-def)
    subgoal for i j
    apply (case-tac ts!i)
    apply (case-tac ts!j)
    apply force
    done
    done
next
  from init have read-only-unowned 𝒮 ts
    by (simp add: initialv-def)
  then show read-only-unowned 𝒮 tsd
    apply (clarsimp simp add: tsd-def read-only-unowned-def)
    subgoal for i
    apply (case-tac ts!i)
    apply force
    done
    done
next
  from init have unowned-shared 𝒮 ts
    by (simp add: initialv-def)
  then
  show unowned-shared 𝒮 tsd
    apply (clarsimp simp add: tsd-def unowned-shared-def)
    apply force
    done
next
  have eq: ((λ(p, is, j, sb, 𝒟, 𝒪, ℛ). (p, is, j, [], 𝒟, 𝒪, ℛ)) ∘
          (λ(p, is, j, sb, 𝒟, 𝒪, ℛ'). (p, is, j, (), 𝒟, 𝒪, Map.empty)))
   = (λ(p, is, j, sb, 𝒟, 𝒪, ℛ'). (p, is, j, [], 𝒟, 𝒪, Map.empty))
    apply (rule ext)
    subgoal for x
    apply (case-tac x)
    apply auto
    done
    done
  from init have initialv-axioms ts valid
    by (simp add: initialv-def)

  then
  show initiald-axioms tsd valid
    apply (clarsimp simp add: tsd-def initialv-axioms-def initiald-axioms-def eq)
    apply (rule conjI)
    apply  clarsimp
           subgoal for i
           apply (case-tac ts!i)
           apply force
           done
    apply clarsimp
    subgoal for i
```

apply (case-tac ts!i)
apply force
done
done
qed

{
fix $ts_d{}'$ m' $\mathcal{S}'$
assume exec: $(ts_d, m, \mathcal{S}) \Rightarrow_d{}^* (ts_d{}', m', \mathcal{S}')$
have safe-free-flowing $(ts_d{}', m', \mathcal{S}')$
proof −
from virtual-simulates-direct-steps [OF exec]
have exec-v: $(ts, m, \mathcal{S}) \Rightarrow_v{}^*$ (remove-rels $ts_d{}'$, m', $\mathcal{S}'$)
by (simp add: rem-ts)
have eq: map (owned ∘
$(\lambda(p, is, j, sb, \mathcal{D}, \mathcal{O}, \mathcal{R}). (p, is, j, (), \mathcal{D}, \mathcal{O}, ())))$
$ts_d{}'$ = map owned $ts_d{}'$
by auto
from exec-v safe
have safe-free-flowing (remove-rels $ts_d{}'$, m', $\mathcal{S}'$)
by (auto simp add: safe-reach-def)
then show ?thesis
by (auto simp add: safe-free-flowing-def remove-rels-def owned-def eq)
qed
}
hence safe′: safe-reach-direct safe-free-flowing $(ts_d, m, \mathcal{S})$
by (simp add: safe-reach-def)

from store-buffer-execution-result-sequential-consistent [OF exec-sb empty′ sim′ init′ safe′]
obtain $ts_d{}'$ $\mathcal{S}'$ **where**
exec-d: $(ts_d, m, \mathcal{S}) \Rightarrow_d{}^* (ts_d{}', m', \mathcal{S}')$ **and** sim-d: $ts_{sb}{}' \sim_d ts_d{}'$
by blast

from virtual-simulates-direct-steps [OF exec-d]
have $(ts, m, \mathcal{S}) \Rightarrow_v{}^*$ (remove-rels $ts_d{}'$, m', $\mathcal{S}'$)
by (simp add: rem-ts)
moreover
from sim-d
have $ts_{sb}{}' \sim_d$ remove-rels $ts_d{}'$
apply (cases)
apply (rule sim-direct-config.intros)
apply (auto simp add: remove-rels-def)
done
ultimately show ?thesis
by auto
qed

A.7 Plug Together the Two Simulations

corollary (**in** xvalid-program) concurrent-direct-steps-simulates-store-buffer-step:
 assumes step-sb: $(ts_{sb},m_{sb},\mathcal{S}_{sb}) \Rightarrow_{sb} (ts_{sb}',m_{sb}',\mathcal{S}_{sb}')$
 assumes sim-h: $ts_{sb} \sim_h ts_{sbh}$
 assumes sim: $(ts_{sbh},m_{sb},\mathcal{S}_{sbh}) \sim (ts,m,\mathcal{S})$
 assumes valid-own: valid-ownership $\mathcal{S}_{sbh}$ ts_{sbh}
 assumes valid-sb-reads: valid-reads m_{sb} ts_{sbh}
 assumes valid-hist: valid-history program-step ts_{sbh}
 assumes valid-sharing: valid-sharing $\mathcal{S}_{sbh}$ ts_{sbh}
 assumes tmps-distinct: tmps-distinct ts_{sbh}
 assumes valid-sops: valid-sops ts_{sbh}
 assumes valid-dd: valid-data-dependency ts_{sbh}
 assumes load-tmps-fresh: load-tmps-fresh ts_{sbh}
 assumes enough-flushs: enough-flushs ts_{sbh}
 assumes valid-program-history: valid-program-history ts_{sbh}
 assumes valid: valid ts_{sbh}
 assumes safe-reach: safe-reach-direct safe-delayed $(ts,m,\mathcal{S})$
 shows $\exists\, ts_{sbh}'\ \mathcal{S}_{sbh}'$.
  $(ts_{sbh},m_{sb},\mathcal{S}_{sbh}) \Rightarrow_{sbh}^* (ts_{sbh}',m_{sb}',\mathcal{S}_{sbh}') \wedge ts_{sb}' \sim_h ts_{sbh}' \wedge$
  valid-ownership $\mathcal{S}_{sbh}'$ ts_{sbh}' $\wedge$ valid-reads m_{sb}' ts_{sbh}' $\wedge$
  valid-history program-step ts_{sbh}' $\wedge$
  valid-sharing $\mathcal{S}_{sbh}'$ ts_{sbh}' $\wedge$ tmps-distinct ts_{sbh}' $\wedge$ valid-data-dependency ts_{sbh}' $\wedge$
  valid-sops ts_{sbh}' $\wedge$ load-tmps-fresh ts_{sbh}' $\wedge$ enough-flushs ts_{sbh}' $\wedge$
  valid-program-history ts_{sbh}' $\wedge$ valid ts_{sbh}' $\wedge$
   $(\exists\, ts'\ \mathcal{S}'\ m'.\ (ts,m,\mathcal{S}) \Rightarrow_d^* (ts',m',\mathcal{S}') \wedge$
   $(ts_{sbh}',m_{sb}',\mathcal{S}_{sbh}') \sim (ts',m',\mathcal{S}'))$
proof −
 from concurrent-history-steps-simulates-store-buffer-step [OF step-sb sim-h]
 obtain ts_{sbh}' $\mathcal{S}_{sbh}'$ **where**
  steps-h: $(ts_{sbh},m_{sb},\mathcal{S}_{sbh}) \Rightarrow_{sbh}^* (ts_{sbh}',m_{sb}',\mathcal{S}_{sbh}')$ **and**
  sim-h′: $ts_{sb}' \sim_h ts_{sbh}'$
  by blast
 moreover
 from concurrent-direct-steps-simulates-store-buffer-history-steps [OF steps-h
 valid-own valid-sb-reads valid-hist valid-sharing tmps-distinct valid-sops valid-dd
 load-tmps-fresh enough-flushs valid-program-history valid sim safe-reach]
 obtain m' ts' $\mathcal{S}'$ **where**
  $(ts,m,\mathcal{S}) \Rightarrow_d^* (ts',m',\mathcal{S}')$ $(ts_{sbh}', m_{sb}',\mathcal{S}_{sbh}') \sim (ts', m', \mathcal{S}')$
  valid-ownership $\mathcal{S}_{sbh}'$ ts_{sbh}' valid-reads m_{sb}' ts_{sbh}' valid-history program-step ts_{sbh}'
  valid-sharing $\mathcal{S}_{sbh}'$ ts_{sbh}' tmps-distinct ts_{sbh}' valid-data-dependency ts_{sbh}'
  valid-sops ts_{sbh}' load-tmps-fresh ts_{sbh}' enough-flushs ts_{sbh}'
  valid-program-history ts_{sbh}' valid ts_{sbh}'
  by blast
 ultimately
 show ?thesis
  by blast
qed

lemma conj-commI: $P \wedge Q \Longrightarrow Q \wedge P$
  by simp
lemma def-to-eq: $P = Q \Longrightarrow P \equiv Q$
  by simp

context xvalid-program
begin

definition
invariant ts $\mathcal{S}$ m $\equiv$
  valid-ownership $\mathcal{S}$ ts $\wedge$ valid-reads m ts $\wedge$ valid-history program-step ts $\wedge$
  valid-sharing $\mathcal{S}$ ts $\wedge$ tmps-distinct ts $\wedge$ valid-data-dependency ts $\wedge$
  valid-sops ts $\wedge$ load-tmps-fresh ts $\wedge$ enough-flushs ts $\wedge$ valid-program-history ts $\wedge$
  valid ts

definition ownership-inv $\equiv$ valid-ownership
definition sharing-inv $\equiv$ valid-sharing
definition temporaries-inv ts $\equiv$ tmps-distinct ts $\wedge$ load-tmps-fresh ts
definition history-inv ts m $\equiv$ valid-history program-step ts $\wedge$ valid-program-history ts $\wedge$ valid-reads m ts
definition data-dependency-inv ts $\equiv$ valid-data-dependency ts $\wedge$ load-tmps-fresh ts $\wedge$ valid-sops ts
definition barrier-inv $\equiv$ enough-flushs

lemma invariant-grouped-def: invariant ts $\mathcal{S}$ m $\equiv$
 ownership-inv $\mathcal{S}$ ts $\wedge$ sharing-inv $\mathcal{S}$ ts $\wedge$ temporaries-inv ts $\wedge$ data-dependency-inv ts $\wedge$ history-inv ts m $\wedge$ barrier-inv ts $\wedge$ valid ts
  apply (rule def-to-eq)
   apply (auto simp add: ownership-inv-def sharing-inv-def barrier-inv-def temporaries-inv-def history-inv-def data-dependency-inv-def invariant-def)
  done

theorem (**in** xvalid-program) simulation$'$:
  assumes step-sb: $(ts_{sb},m_{sb},\mathcal{S}_{sb}) \Rightarrow_{sbh} (ts_{sb}',m_{sb}',\mathcal{S}_{sb}')$
  assumes sim: $(ts_{sb},m_{sb},\mathcal{S}_{sb}) \sim (ts,m,\mathcal{S})$
  assumes inv: invariant ts_{sb} $\mathcal{S}_{sb}$ m_{sb}
  assumes safe-reach: safe-reach-direct safe-delayed $(ts,m,\mathcal{S})$
  shows invariant ts_{sb}' $\mathcal{S}_{sb}'$ m_{sb}' $\wedge$
      $(\exists ts'\ \mathcal{S}'\ m'.\ (ts,m,\mathcal{S}) \Rightarrow_d{}^* (ts',m',\mathcal{S}') \wedge (ts_{sb}',m_{sb}',\mathcal{S}_{sb}') \sim (ts',m',\mathcal{S}'))$
  using inv sim safe-reach
  apply (unfold invariant-def)
  apply (simp only: conj-assoc)
  apply (rule concurrent-direct-steps-simulates-store-buffer-history-step [OF step-sb])
  apply simp-all
  done

lemmas (**in** xvalid-program) simulation = conj-commI [OF simulation$'$]
end

end

A.8 PIMP

theory PIMP
imports ReduceStoreBufferSimulation
begin

datatype expr = Const val | Mem bool addr | Tmp sop
| Unop val ⇒ val expr
| Binop val ⇒ val ⇒ val expr expr

datatype stmt =
Skip
| Assign bool expr expr tmps ⇒ owns tmps ⇒ owns tmps ⇒ owns tmps ⇒ owns
| CAS expr expr expr tmps ⇒ owns tmps ⇒ owns tmps ⇒ owns tmps ⇒ owns
| Seq stmt stmt
| Cond expr stmt stmt
| While expr stmt

| SGhost tmps ⇒ owns tmps ⇒ owns tmps ⇒ owns tmps ⇒ owns
| SFence

primrec used-tmps:: expr ⇒ nat — number of temporaries used
where
used-tmps (Const v) = 0
| used-tmps (Mem volatile addr) = 1
| used-tmps (Tmp sop) = 0
| used-tmps (Unop f e) = used-tmps e
| used-tmps (Binop f e_1 e_2) = used-tmps e_1 + used-tmps e_2

primrec issue-expr:: tmp ⇒ expr ⇒ instr list — load operations
where
issue-expr t (Const v) = []
|issue-expr t (Mem volatile a) = [Read volatile a t]
|issue-expr t (Tmp sop) = []
|issue-expr t (Unop f e) = issue-expr t e
|issue-expr t (Binop f e_1 e_2) = issue-expr t e_1 @ issue-expr (t + (used-tmps e_1)) e_2

primrec eval-expr:: tmp ⇒ expr ⇒ sop — calculate result
where
eval-expr t (Const v) = ({},λj. v)
|eval-expr t (Mem volatile a) = ({t},λj. the (j t))
|eval-expr t (Tmp sop) = sop

— trick to enforce sop to be sensible in the current context, without having to include wellformedness constraints
|eval-expr t (Unop f e) = (let (D,f_e) = eval-expr t e in $(D,\lambda j.\ f\ (f_e\ j)))$
|eval-expr t (Binop f e_1 e_2) = (let (D_1,f_1) = eval-expr t e_1;
(D_2,f_2) = eval-expr (t + (used-tmps e_1)) e_2
in $(D_1 \cup D_2,\lambda j.\ f\ (f_1\ j)\ (f_2\ j)))$

primrec valid-sops-expr:: nat $\Rightarrow$ expr $\Rightarrow$ bool
where
valid-sops-expr t (Const v) = True
|valid-sops-expr t (Mem volatile a) = True
|valid-sops-expr t (Tmp sop) = $((\forall t' \in$ fst sop. $t' < t) \wedge$ valid-sop sop)
|valid-sops-expr t (Unop f e) = valid-sops-expr t e
|valid-sops-expr t (Binop f e_1 e_2) = (valid-sops-expr t e_1 $\wedge$ valid-sops-expr t e_2)

primrec valid-sops-stmt:: nat $\Rightarrow$ stmt $\Rightarrow$ bool
where
valid-sops-stmt t Skip = True
|valid-sops-stmt t (Assign volatile a e A L R W) = (valid-sops-expr t a $\wedge$ valid-sops-expr t e)
|valid-sops-stmt t (CAS a c_e s_e A L R W) = (valid-sops-expr t a $\wedge$ valid-sops-expr t c_e $\wedge$
valid-sops-expr t s_e)
|valid-sops-stmt t (Seq s_1 s_2) = (valid-sops-stmt t s_1 $\wedge$ valid-sops-stmt t s_2)
|valid-sops-stmt t (Cond e s_1 s_2) = (valid-sops-expr t e $\wedge$ valid-sops-stmt t s_1 $\wedge$ valid-sops-stmt t s_2)
|valid-sops-stmt t (While e s) = (valid-sops-expr t e $\wedge$ valid-sops-stmt t s)
|valid-sops-stmt t (SGhost A L R W) = True
|valid-sops-stmt t SFence = True

type-synonym stmt-config = stmt $\times$ nat
consts isTrue:: val $\Rightarrow$ bool

inductive stmt-step:: tmps $\Rightarrow$ stmt-config $\Rightarrow$ stmt-config $\times$ instrs $\Rightarrow$ bool
(‹-$\vdash$ - $\rightarrow_s$ -› [60,60,60] 100)
for j
where

AssignAddr:
$\forall$sop. a $\neq$ Tmp sop $\Longrightarrow$
j$\vdash$ (Assign volatile a e A L R W, t) $\rightarrow_s$
((Assign volatile (Tmp (eval-expr t a)) e A L R W, t + used-tmps a), issue-expr t a)

| Assign:
D $\subseteq$ dom j $\Longrightarrow$
j$\vdash$ (Assign volatile (Tmp (D,a)) e A L R W, t) $\rightarrow_s$
((Skip, t + used-tmps e),
issue-expr t e@[Write volatile (a j) (eval-expr t e) (A j) (L j) (R j) (W j)])

| CASAddr:
$\forall$sop. a $\neq$ Tmp sop $\Longrightarrow$
j$\vdash$ (CAS a c_e s_e A L R W, t) $\rightarrow_s$
((CAS (Tmp (eval-expr t a)) c_e s_e A L R W, t + used-tmps a), issue-expr t a)

| CASComp:
$\forall$sop. c_e $\neq$ Tmp sop $\Longrightarrow$
j$\vdash$ (CAS (Tmp (D_a,a)) c_e s_e A L R W, t) $\rightarrow_s$
((CAS (Tmp (D_a,a)) (Tmp (eval-expr t c_e)) s_e A L R W, t + used-tmps c_e), issue-expr t c_e)

| CAS:
$[\![$$D_a$ $\subseteq$ dom j; D_c $\subseteq$ dom j; eval-expr t s_e = (D,f) $]\!]$
$\Longrightarrow$
j$\vdash$ (CAS (Tmp (D_a,a)) (Tmp (D_c,c)) s_e A L R W, t) $\rightarrow_s$
((Skip, Suc (t + used-tmps s_e)), issue-expr t s_e@
[RMW (a j) (t + used-tmps s_e) (D,f) (λj. the (j (t + used-tmps s_e)) = c j) ($\lambda$$v_1$ v_2. v_1)
(A j) (L j) (R j) (W j)])

| Seq:
j$\vdash$ (s_1, t) $\rightarrow_s$ ((s_1', t'), is)
$\Longrightarrow$
j$\vdash$ (Seq s_1 s_2, t) $\rightarrow_s$ ((Seq s_1' s_2, t'),is)

| SeqSkip:
j$\vdash$ (Seq Skip s_2, t) $\rightarrow_s$ ((s_2, t), [])

| Cond:
$\forall$sop. e $\neq$ Tmp sop
$\Longrightarrow$
j$\vdash$ (Cond e s_1 s_2, t) $\rightarrow_s$
((Cond (Tmp (eval-expr t e)) s_1 s_2, t + used-tmps e), issue-expr t e)

| CondTrue:
$[\![$D $\subseteq$ dom j; isTrue (e j)$]\!]$
$\Longrightarrow$
j$\vdash$ (Cond (Tmp (D,e)) s_1 s_2, t) $\rightarrow_s$ ((s_1, t),[])

| CondFalse:
$[\![$D $\subseteq$ dom j; $\neg$ isTrue (e j)$]\!]$
$\Longrightarrow$
j$\vdash$ (Cond (Tmp (D,e)) s_1 s_2, t) $\rightarrow_s$ ((s_2, t),[])

| While:
j$\vdash$ (While e s, t) $\rightarrow_s$

((Cond e (Seq s (While e s)) Skip, t),[])

| SGhost:
j⊢ (SGhost A L R W, t) $\rightarrow_{\mathsf{s}}$ ((Skip, t),[Ghost (A j) (L j) (R j) (W j)])

| SFence:
j⊢ (SFence, t) $\rightarrow_{\mathsf{s}}$ ((Skip, t),[Fence])

inductive-cases stmt-step-cases [cases set]:
j⊢ (Skip, t) $\rightarrow_{\mathsf{s}}$ c
j⊢ (Assign volatile a e A L R W, t) $\rightarrow_{\mathsf{s}}$ c
j⊢ (CAS a c_{e} s_{e} A L R W, t) $\rightarrow_{\mathsf{s}}$ c
j⊢ (Seq s_1 s_2, t) $\rightarrow_{\mathsf{s}}$ c
j⊢ (Cond e s_1 s_2, t) $\rightarrow_{\mathsf{s}}$ c
j⊢ (While e s, t) $\rightarrow_{\mathsf{s}}$ c
j⊢ (SGhost A L R W, t) $\rightarrow_{\mathsf{s}}$ c
j⊢ (SFence, t) $\rightarrow_{\mathsf{s}}$ c

lemma valid-sops-expr-mono: $\bigwedge$t t′. valid-sops-expr t e $\Longrightarrow$ t $\leq$ t′ $\Longrightarrow$ valid-sops-expr t′ e
  by (induct e) auto

lemma valid-sops-stmt-mono: $\bigwedge$t t′. valid-sops-stmt t s $\Longrightarrow$ t $\leq$ t′ $\Longrightarrow$ valid-sops-stmt t′ s
  by (induct s) (auto intro: valid-sops-expr-mono)

lemma valid-sops-expr-valid-sop: $\bigwedge$t. valid-sops-expr t e $\Longrightarrow$ valid-sop (eval-expr t e)
proof (induct e)
  case (Unop f e)
  then obtain valid-sops-expr t e
    by simp
  from Unop.hyps [OF this]
  have vs: valid-sop (eval-expr t e)
    by simp
  obtain D g **where** eval-e: eval-expr t e = (D,g)
    by (cases eval-expr t e)

  interpret valid-sop (D,g)
    using vs eval-e
    by simp

  show ?case
    apply (clarsimp simp add: Let-def valid-sop-def eval-e)
    apply (drule valid-sop [OF refl])
    apply simp
    done
next
  case (Binop f e_1 e_2)
  then obtain v1: valid-sops-expr t e_1 **and** v2: valid-sops-expr t e_2
    by simp

with Binop.hyps (1) [of t] Binop.hyps (2) [of (t + used-tmps e_1)]
valid-sops-expr-mono [OF v2, of (t + used-tmps e_1)]
obtain vs1: valid-sop (eval-expr t e_1) **and** vs2: valid-sop (eval-expr (t + used-tmps e_1) e_2)
by auto
obtain D_1 g_1 **where** eval-e_1: eval-expr t e_1 = (D_1,g_1)
by (cases eval-expr t e_1)
obtain D_2 g_2 **where** eval-e_2: eval-expr (t + used-tmps e_1) e_2 = (D_2,g_2)
by (cases eval-expr (t + used-tmps e_1) e_2)
interpret vs1: valid-sop (D_1,g_1)
using vs1 eval-e_1 **by** auto
interpret vs2: valid-sop (D_2,g_2)
using vs2 eval-e_2 **by** auto
{
fix j:: nat⇒val option
assume D1: $D_1 \subseteq$ dom j
assume D2: $D_2 \subseteq$ dom j
have f (g_1 j) (g_2 j) = f (g_1 (j |‘ ($D_1 \cup D_2$))) (g_2 (j |‘ ($D_1 \cup D_2$)))
proof −
from vs1.valid-sop [OF refl D1]
have g_1 j = g_1 (j |‘ D_1)**.**
also
from D1 **have** D1′: $D_1 \subseteq$ dom (j |‘ ($D_1 \cup D_2$))
by auto
have j |‘ ($D_1 \cup D_2$) |‘ D_1 = j |‘ D_1
apply (rule ext)
apply (auto simp add: restrict-map-def)
done
with vs1.valid-sop [OF refl D1′]
have g_1 (j |‘ D_1) = g_1 (j |‘ ($D_1 \cup D_2$))
by auto
finally have g1: g_1 (j |‘ ($D_1 \cup D_2$)) = g_1 j
by simp

from vs2.valid-sop [OF refl D2]
have g_2 j = g_2 (j |‘ D_2)**.**
also
from D2 **have** D2′: $D_2 \subseteq$ dom (j |‘ ($D_1 \cup D_2$))
by auto
have j |‘ ($D_1 \cup D_2$) |‘ D_2 = j |‘ D_2
apply (rule ext)
apply (auto simp add: restrict-map-def)
done
with vs2.valid-sop [OF refl D2′]
have g_2 (j |‘ D_2) = g_2 (j |‘ ($D_1 \cup D_2$))
by auto
finally have g2: g_2 (j |‘ ($D_1 \cup D_2$)) = g_2 j
by simp

from g1 g2 **show** ?thesis **by** simp

qed
}

note lem=this
show ?case
apply (clarsimp simp add: Let-def valid-sop-def eval-e_1 eval-e_2)
apply (rule lem)
by auto
qed (auto simp add: valid-sop-def)

lemma valid-sops-expr-eval-expr-in-range:
$\bigwedge$t. valid-sops-expr t e $\Longrightarrow$ $\forall$t′ $\in$ fst (eval-expr t e). t′ < t + used-tmps e
proof (induct e)
case (Unop f e)
thus ?case
apply (cases eval-expr t e)
apply auto
done
next
case (Binop f e_1 e_2)
then obtain v1: valid-sops-expr t e_1 **and** v2: valid-sops-expr t e_2
by simp
from valid-sops-expr-mono [OF v2]
have v2′: valid-sops-expr (t + used-tmps e_1) e_2
by auto
from Binop.hyps (1) [OF v1] Binop.hyps (2) [OF v2′]
show ?case
apply (cases eval-expr t e_1)
apply (cases eval-expr (t + used-tmps e_1) e_2)
apply auto
done
qed auto

lemma stmt-step-tmps-count-mono:
assumes step: j$\vdash$ (s,t) $\rightarrow_s$ ((s′,t′),is)
shows t $\leq$ t′
using step
by (induct x==(s,t) y==((s′,t′),is) arbitrary: s t s′ t′ is rule: stmt-step.induct) force+

lemma valid-sops-stmt-invariant:
assumes step: j$\vdash$ (s,t) $\rightarrow_s$ ((s′,t′),is)
shows valid-sops-stmt t s $\Longrightarrow$ valid-sops-stmt t′ s′
using step
proof (induct x==(s,t) y==((s′,t′),is) arbitrary: s t s′ t′ is rule: stmt-step.induct)
case AssignAddr **thus** ?case **by**
(force simp add: valid-sops-expr-valid-sop intro: valid-sops-stmt-mono valid-sops-expr-mono

dest: valid-sops-expr-eval-expr-in-range)
next
case Assign **thus** ?case **by** simp
next
case CASAddr **thus** ?case **by**
(force simp add: valid-sops-expr-valid-sop intro: valid-sops-stmt-mono valid-sops-expr-mono
dest: valid-sops-expr-eval-expr-in-range)
next
case CASComp **thus** ?case **by**
(force simp add: valid-sops-expr-valid-sop intro: valid-sops-stmt-mono valid-sops-expr-mono
dest: valid-sops-expr-eval-expr-in-range)
next
case CAS **thus** ?case **by** simp
next
case Seq **thus** ?case **by** (force intro: valid-sops-stmt-mono dest: stmt-step-tmps-count-mono)
next
case SeqSkip **thus** ?case **by** auto
next
case Cond **thus** ?case
by (fastforce simp add: valid-sops-expr-valid-sop intro: valid-sops-stmt-mono
dest: valid-sops-expr-eval-expr-in-range)
next
case CondTrue **thus** ?case **by** force
next
case CondFalse **thus** ?case **by** force
next
case While **thus** ?case **by** auto
next
case SGhost **thus** ?case **by** simp
next
case SFence **thus** ?case **by** simp
qed

lemma map-le-restrict-map-eq: $m_1 \subseteq_m m_2 \Longrightarrow D \subseteq \text{dom } m_1 \Longrightarrow m_2 |\text{` } D = m_1 |\text{` } D$
apply (rule ext)
apply (force simp add: restrict-map-def map-le-def)
done

lemma sbh-step-preserves-load-tmps-bound:
assumes step: $(is,\mathcal{O},\mathcal{D},j,sb,\mathcal{S},m) \rightarrow_{\mathsf{sbh}} (is',\mathcal{O}',\mathcal{D}',j',sb',\mathcal{S}',m')$
assumes less: $\forall i \in$ load-tmps is. $i < n$
shows $\forall i \in$ load-tmps is$'$. $i < n$
using step less
by cases auto

lemma sbh-step-preserves-read-tmps-bound:
  assumes step: (is,j,sb,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$ (is′,j′,sb′,m′,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{S}'$)
  assumes less-is: $\forall$ i $\in$ load-tmps is. i $<$ n
  assumes less-sb: $\forall$ i $\in$ read-tmps sb. i $<$ n
  shows $\forall$ i $\in$ read-tmps sb′. i $<$ n
  using step less-is less-sb
  by cases (auto simp add: read-tmps-append)

lemma sbh-step-preserves-tmps-bound:
  assumes step: (is,j,sb,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$ (is′,j′,sb′,m′,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{S}'$)
  assumes less-dom: $\forall$ i $\in$ dom j. i $<$ n
  assumes less-is: $\forall$ i $\in$ load-tmps is. i $<$ n
  shows $\forall$ i $\in$ dom j′. i $<$ n
  using step less-dom less-is
  by cases (auto simp add: read-tmps-append)

lemma flush-step-preserves-read-tmps:
  assumes step: (m,sb,$\mathcal{O}$) $\rightarrow_{\mathsf{f}}$ (m′,sb′,$\mathcal{O}'$)
  assumes less-sb: $\forall$ i $\in$ read-tmps sb. i $<$ n
  shows $\forall$ i $\in$ read-tmps sb′. i $<$ n
  using step less-sb
  by cases (auto simp add: read-tmps-append)

lemma flush-step-preserves-write-sops:
  assumes step: (m,sb,$\mathcal{O}$) $\rightarrow_{\mathsf{f}}$ (m′,sb′,$\mathcal{O}'$)
  assumes less-sb: $\forall$i$\in\bigcup$(fst ‘ write-sops sb). i $<$ t
  shows $\forall$i$\in\bigcup$(fst ‘ write-sops sb′). i $<$ t
  using step less-sb
  by cases (auto simp add: read-tmps-append)

lemma issue-expr-load-tmps-range′:
  $\bigwedge$t. load-tmps (issue-expr t e) = {i. t $\leq$ i $\wedge$ i $<$ t + used-tmps e}
apply (induct e)
apply (force simp add: load-tmps-append)+
done

lemma issue-expr-load-tmps-range:
  $\bigwedge$t. $\forall$ i $\in$ load-tmps (issue-expr t e). t $\leq$ i $\wedge$ i $<$ t + (used-tmps e)
by (auto simp add: issue-expr-load-tmps-range′)

lemma stmt-step-load-tmps-range′:
  assumes step: j$\vdash$ (s, t) $\rightarrow_{\mathsf{s}}$ ((s′, t′),is)
  shows load-tmps is = {i. t $\leq$ i $\wedge$ i $<$ t′}
  using step
  apply (induct x==(s,t) y==((s′,t′),is) arbitrary: s t s′ t′ is rule: stmt-step.induct)
  apply (force simp add: load-tmps-append simp add: issue-expr-load-tmps-range′)+
  done

lemma stmt-step-load-tmps-range:
  assumes step: j⊢ (s, t) $\rightarrow_{\mathsf{s}}$ ((s′, t′),is)
  shows ∀ i ∈ load-tmps is. t ≤ i ∧ i < t′
using stmt-step-load-tmps-range′ [OF step]
by auto

lemma distinct-load-tmps-issue-expr: ⋀t. distinct-load-tmps (issue-expr t e)
  apply (induct e)
   apply (auto simp add: distinct-load-tmps-append dest!: issue-expr-load-tmps-range [rule-format])
  done

lemma max-used-load-tmps: t + used-tmps e ∉ load-tmps (issue-expr t e)
proof −
  from issue-expr-load-tmps-range [rule-format, of t+used-tmps e]
  show ?thesis
    by auto
qed

lemma stmt-step-distinct-load-tmps:
  assumes step: j⊢ (s, t) $\rightarrow_{\mathsf{s}}$ ((s′, t′),is)
  shows distinct-load-tmps is
  using step
  apply (induct x==(s,t) y==((s′,t′),is) arbitrary: s t s′ t′ is rule: stmt-step.induct)
    apply (force simp add: distinct-load-tmps-append distinct-load-tmps-issue-expr max-used-load-tmps)+
  done

lemma store-sops-issue-expr [simp]: ⋀t. store-sops (issue-expr t e) = {}
  apply (induct e)
  apply (auto simp add: store-sops-append)
  done

lemma stmt-step-data-store-sops-range:
  assumes step: j⊢ (s, t) $\rightarrow_{\mathsf{s}}$ ((s′, t′),is)
  assumes valid: valid-sops-stmt t s
  shows ∀ (D,f) ∈ store-sops is. ∀ i ∈ D. i < t′
using step valid
proof (induct x==(s,t) y==((s′,t′),is) arbitrary: s t s′ t′ is rule: stmt-step.induct)
  case AssignAddr
  thus ?case
    by auto
next
  case (Assign D volatile a e)
  thus ?case
    apply (cases eval-expr t e)

apply (auto simp add: store-sops-append intro: valid-sops-expr-eval-expr-in-range [rule-format])
done
next
case CASAddr
thus ?case
by auto
next
case CASComp
thus ?case
by auto
next
case (CAS - - D f a A L R)
thus ?case
by (fastforce simp add: store-sops-append dest: valid-sops-expr-eval-expr-in-range [rule-format])
next
case Seq
thus ?case
by (force intro: valid-sops-stmt-mono)
next
case SeqSkip **thus** ?case **by** simp
next
case Cond **thus** ?case
by auto
next
case CondTrue **thus** ?case **by** auto
next
case CondFalse **thus** ?case **by** auto
next
case While **thus** ?case **by** auto
next
case SGhost **thus** ?case **by** auto
next
case SFence **thus** ?case **by** auto
qed

lemma sbh-step-distinct-load-tmps-prog-step:
assumes step: $j \vdash (s,t) \rightarrow_{s} ((s',t'),is')$
assumes load-tmps-le: $\forall i \in$ load-tmps is. $i < t$
assumes read-tmps-le: $\forall i \in$ read-tmps sb. $i < t$
shows distinct-load-tmps is' $\wedge$ (load-tmps is' $\cap$ load-tmps is = {}) $\wedge$
(load-tmps is' $\cap$ read-tmps sb) = {}
proof −
from stmt-step-load-tmps-range [OF step] stmt-step-distinct-load-tmps [OF step]
load-tmps-le read-tmps-le
show ?thesis
by force
qed

lemma data-dependency-consistent-instrs-issue-expr:
  $\bigwedge$t T. data-dependency-consistent-instrs T (issue-expr t e)
  apply (induct e)
  apply (auto simp add: data-dependency-consistent-instrs-append
    dest!: issue-expr-load-tmps-range [rule-format]
)
  done

lemma dom-eval-expr:
  $\bigwedge$t. $[\![$valid-sops-expr t e; x $\in$ fst (eval-expr t e)$]\!] \Longrightarrow$ x $\in$ {i. i < t} $\cup$ load-tmps (issue-expr t e)
proof (induct e)
  case Const **thus** ?case **by** simp
next
  case Mem **thus** ?case **by** simp
next
  case Tmp **thus** ?case **by** simp
next
  case (Unop f e)
  thus ?case
    by (cases eval-expr t e) auto
next
  case (Binop f e1 e2)
  then obtain valid1: valid-sops-expr t e1 **and** valid2: valid-sops-expr t e2
    by auto
  from valid-sops-expr-mono [OF valid2] **have** valid2$'$: valid-sops-expr (t+used-tmps e1) e2
    by auto

  from Binop.hyps (1) [OF valid1] Binop.hyps (2) [OF valid2$'$] Binop.prems
  show ?case
    apply (case-tac eval-expr t e1)
    apply (case-tac eval-expr (t+used-tmps e1) e2)
    apply (auto simp add: load-tmps-append issue-expr-load-tmps-range$'$)
    done
qed

lemma Cond-not-s_1: $s_1 \neq$ Cond e s_1 s_2
  by (induct s_1) auto

lemma Cond-not-s_2: $s_2 \neq$ Cond e s_1 s_2
  by (induct s_2) auto

lemma Seq-not-s_1: $s_1 \neq$ Seq s_1 s_2
  by (induct s_1) auto

lemma Seq-not-s_2: $s_2 \neq$ Seq s_1 s_2
  by (induct s_2) auto

lemma prog-step-progress:
  assumes step: $j\vdash(s,t) \rightarrow_s ((s',t'),is)$
  shows $(s',t') \neq (s,t) \vee is \neq []$
using step
proof (induct $x==(s,t)$ $y==((s',t'),is)$ arbitrary: s t s' t' is rule: stmt-step.induct)
  case (AssignAddr a - - - - - - t) **thus** ?case
    by (cases eval-expr t a) auto
next
  case Assign **thus** ?case **by** auto
next
  case (CASAddr a - - - - - - t) **thus** ?case **by** (cases eval-expr t a) auto
next
  case (CASComp c_e - - - - - - - t) **thus** ?case **by** (cases eval-expr t c_e) auto
next
  case CAS **thus** ?case **by** auto
next
  case (Cond e - - t) **thus** ?case **by** (cases eval-expr t e) auto
next
  case CondTrue **thus** ?case **using** Cond-not-s_1 **by** auto
next
  case CondFalse **thus** ?case **using** Cond-not-s_2 **by** auto
next
  case Seq **thus** ?case **by** force
next
  case SeqSkip **thus** ?case **using** Seq-not-s_2 **by** auto
next
  case While **thus** ?case **by** auto
next
  case SGhost **thus** ?case **by** auto
next
  case SFence **thus** ?case **by** auto
qed

lemma stmt-step-data-dependency-consistent-instrs:
  assumes step: $j\vdash$ (s, t) $\rightarrow_s$ ((s', t'),is)
  assumes valid: valid-sops-stmt t s
  shows data-dependency-consistent-instrs ({i. i < t}) is
  using step valid
proof (induct $x==(s,t)$ $y==((s',t'),is)$ arbitrary: s t s' t' is T rule: stmt-step.induct)
  case AssignAddr
  thus ?case
    by (fastforce simp add: simp add: data-dependency-consistent-instrs-append
    data-dependency-consistent-instrs-issue-expr load-tmps-append
    dest: dom-eval-expr)
next
  case Assign
  thus ?case
    by (fastforce simp add: simp add: data-dependency-consistent-instrs-append
    data-dependency-consistent-instrs-issue-expr load-tmps-append

dest: dom-eval-expr)
next
case CASAddr
thus ?case
by (fastforce simp add: simp add: data-dependency-consistent-instrs-append
data-dependency-consistent-instrs-issue-expr load-tmps-append
dest: dom-eval-expr)
next
case CASComp
thus ?case
by (fastforce simp add: simp add: data-dependency-consistent-instrs-append
data-dependency-consistent-instrs-issue-expr load-tmps-append
dest: dom-eval-expr)
next
case CAS
thus ?case
by (fastforce simp add: simp add: data-dependency-consistent-instrs-append
data-dependency-consistent-instrs-issue-expr load-tmps-append
dest: dom-eval-expr)
next
case Seq
thus ?case
by (fastforce simp add: simp add: data-dependency-consistent-instrs-append)
next
case SeqSkip **thus** ?case **by** auto
next
case Cond
thus ?case
by (fastforce simp add: simp add: data-dependency-consistent-instrs-append
data-dependency-consistent-instrs-issue-expr load-tmps-append
dest: dom-eval-expr)
next
case CondTrue **thus** ?case **by** auto
next
case CondFalse **thus** ?case **by** auto
next
case While
thus ?case **by** auto
next
case SGhost **thus** ?case **by** auto
next
case SFence **thus** ?case **by** auto
qed

lemma sbh-valid-data-dependency-prog-step:
assumes step: $j \vdash (s,t) \rightarrow_s ((s',t'),is')$
assumes store-sops-le: $\forall i \in \bigcup(fst \text{ ` } store\text{-}sops\ is).\ i < t$
assumes write-sops-le: $\forall i \in \bigcup(fst \text{ ` } write\text{-}sops\ sb).\ i < t$

assumes valid: valid-sops-stmt t s
shows data-dependency-consistent-instrs ({i. i < t}) is′ ∧
load-tmps is′ ∩ ⋃(fst ‘ store-sops is) = {} ∧
load-tmps is′ ∩ ⋃(fst ‘ write-sops sb) = {}
proof –
from stmt-step-data-dependency-consistent-instrs [OF step valid]
stmt-step-load-tmps-range [OF step]
store-sops-le write-sops-le
show ?thesis
by fastforce
qed

lemma sbh-load-tmps-fresh-prog-step:
assumes step: j⊢(s,t) $\rightarrow_{s}$ ((s′,t′),is′)
assumes tmps-le: ∀i ∈ dom j. i < t
shows load-tmps is′ ∩ dom j = {}
proof –
from stmt-step-load-tmps-range [OF step] tmps-le
show ?thesis
apply auto
subgoal for x
apply (drule-tac x=x **in** bspec)
apply assumption
apply (drule-tac x=x **in** bspec)
apply fastforce
apply simp
done
done
qed

lemma sbh-valid-sops-prog-step:
assumes step: j⊢(s,t) $\rightarrow_{s}$ ((s′,t′),is)
assumes valid: valid-sops-stmt t s
shows ∀sop∈store-sops is. valid-sop sop
using step valid
proof (induct x==(s,t) y==((s′,t′),is) arbitrary: s t s′ t′ is rule: stmt-step.induct)
case AssignAddr
thus ?case **by** auto
next
case Assign
thus ?case
by (auto simp add: store-sops-append valid-sops-expr-valid-sop)
next
case CASAddr
thus ?case **by** auto
next
case CASComp
thus ?case **by** auto
next
case CAS

thus ?case
by (fastforce simp add: store-sops-append dest: valid-sops-expr-valid-sop)
next
case Seq
thus ?case
by (force intro: valid-sops-stmt-mono)
next
case SeqSkip **thus** ?case **by** simp
next
case Cond **thus** ?case
by auto
next
case CondTrue **thus** ?case **by** auto
next
case CondFalse **thus** ?case **by** auto
next
case While **thus** ?case **by** auto
next
case SGhost **thus** ?case **by** auto
next
case SFence **thus** ?case **by** auto
qed

primrec prog-configs:: $'$a memref list $\Rightarrow$ $'$a set
where
prog-configs [] = {}
|prog-configs (x#xs) = (case x of
Prog$_{\mathsf{sb}}$ p p$'$ is $\Rightarrow$ {p,p$'$} $\cup$ prog-configs xs
| - $\Rightarrow$ prog-configs xs)

lemma prog-configs-append: $\bigwedge$ys. prog-configs (xs@ys) = prog-configs xs $\cup$ prog-configs ys
by (induct xs) (auto split: memref.splits)

lemma prog-configs-in1: Prog$_{\mathsf{sb}}$ p_1 p_2 is $\in$ set xs $\Longrightarrow$ p_1 $\in$ prog-configs xs
by (induct xs) (auto split: memref.splits)

lemma prog-configs-in2: Prog$_{\mathsf{sb}}$ p_1 p_2 is $\in$ set xs $\Longrightarrow$ p_2 $\in$ prog-configs xs
by (induct xs) (auto split: memref.splits)

lemma prog-configs-mono: $\bigwedge$ys. set xs $\subseteq$ set ys $\Longrightarrow$ prog-configs xs $\subseteq$ prog-configs ys
by (induct xs) (auto split: memref.splits simp add: prog-configs-append prog-configs-in1 prog-configs-in2)

locale separated-tmps =
fixes ts
assumes valid-sops-stmt: [[i < length ts; ts!i = ((s,t),is,j,sb,$\mathcal{D}$,$\mathcal{O}$)]]
$\Longrightarrow$ valid-sops-stmt t s
assumes valid-sops-stmt-sb: [[i < length ts; ts!i = ((s,t),is,j,sb,$\mathcal{D}$,$\mathcal{O}$); (s$'$,t$'$) $\in$ prog-configs sb]]

$\Longrightarrow$ valid-sops-stmt t′ s′
assumes load-tmps-le: $[\![$i < length ts; ts!i = ((s,t),is,j,sb,$\mathcal{D}$,$\mathcal{O}$)$]\!]$
$\Longrightarrow$ $\forall$ i $\in$ load-tmps is. i < t
assumes read-tmps-le: $[\![$i < length ts; ts!i = ((s,t),is,j,sb,$\mathcal{D}$,$\mathcal{O}$)$]\!]$
$\Longrightarrow$ $\forall$ i $\in$ read-tmps sb. i < t
assumes store-sops-le: $[\![$i < length ts; ts!i = ((s,t),is,j,sb,$\mathcal{D}$,$\mathcal{O}$)$]\!]$
$\Longrightarrow$ $\forall$ i $\in$ $\bigcup$(fst ‘ store-sops is). i < t
assumes write-sops-le: $[\![$i < length ts; ts!i = ((s,t),is,j,sb,$\mathcal{D}$,$\mathcal{O}$)$]\!]$
$\Longrightarrow$ $\forall$ i $\in$ $\bigcup$(fst ‘ write-sops sb). i < t
assumes tmps-le: $[\![$i < length ts; ts!i = ((s,t),is,j,sb,$\mathcal{D}$,$\mathcal{O}$)$]\!]$
$\Longrightarrow$ dom j $\cup$ load-tmps is = {i. i < t}

lemma (**in** separated-tmps)
tmps-le′:
assumes i-bound: i < length ts
assumes ts-i: ts!i = ((s,t),is,j,sb,$\mathcal{D}$,$\mathcal{O}$)
shows $\forall$ i $\in$ dom j. i < t
using tmps-le [OF i-bound ts-i] **by** auto

lemma (**in** separated-tmps) separated-tmps-nth-update:
$[\![$i < length ts; valid-sops-stmt t s; $\forall$ (s′,t′) $\in$ prog-configs sb. valid-sops-stmt t′ s′;
$\forall$ i $\in$ load-tmps is. i < t;$\forall$ i $\in$ read-tmps sb. i < t;
$\forall$ i $\in$ $\bigcup$(fst ‘ store-sops is). i < t; $\forall$ i $\in$ $\bigcup$(fst ‘ write-sops sb). i < t; dom j $\cup$ load-tmps is = {i. i < t}$]\!]$
$\Longrightarrow$
separated-tmps (ts[i:=((s,t),is,j,sb,$\mathcal{D}$,$\mathcal{O}$)])
apply (unfold-locales)
apply (force intro: valid-sops-stmt simp add: nth-list-update split: if-split-asm)
apply (fastforce intro: valid-sops-stmt-sb simp add: nth-list-update split: if-split-asm)
apply (fastforce intro: load-tmps-le [rule-format] simp add: nth-list-update split: if-split-asm)
apply (fastforce intro: read-tmps-le [rule-format] simp add: nth-list-update split: if-split-asm)
apply (fastforce intro: store-sops-le [rule-format] simp add: nth-list-update split: if-split-asm)
apply (fastforce intro: write-sops-le [rule-format] simp add: nth-list-update split: if-split-asm)
apply (fastforce dest: tmps-le [rule-format] simp add: nth-list-update split: if-split-asm)
done

lemma hd-prog-app-in-first: $\bigwedge$ys. Prog$_{\mathsf{sb}}$ p p′ is $\in$ set xs $\Longrightarrow$ hd-prog q (xs @ ys) = hd-prog q xs
by (induct xs) (auto split: memref.splits)

lemma hd-prog-app-in-eq: $\bigwedge$ys. Prog$_{\mathsf{sb}}$ p p′ is $\in$ set xs $\Longrightarrow$ hd-prog q xs = hd-prog x xs
by (induct xs) (auto split: memref.splits)

lemma hd-prog-app-notin-first: $\bigwedge$ys. $\forall$p p′ is. Prog$_{\mathsf{sb}}$ p p′ is $\notin$ set xs $\Longrightarrow$ hd-prog q (xs @ ys) = hd-prog q ys
  by (induct xs) (auto split: memref.splits)

lemma union-eq-subsetD: A $\cup$ B = C $\Longrightarrow$ A $\cup$ B $\subseteq$ C $\wedge$ C $\subseteq$ A $\cup$ B
  by auto

lemma prog-step-preserves-separated-tmps:
  assumes i-bound: i < length ts
  assumes ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$)
  assumes prog-step: j$\vdash$ p $\rightarrow_{\mathsf{s}}$ (p′, is′)
  assumes sep: separated-tmps ts
  shows separated-tmps
      (ts [i:=(p′,is@is′,j,sb@[Prog$_{\mathsf{sb}}$ p p′ is′],$\mathcal{D}$,$\mathcal{O}$)])
proof −
  obtain s t **where** p: p=(s,t) **by** (cases p)
  obtain s′ t′ **where** p′: p′=(s′,t′) **by** (cases p′)
  note ts-i = ts-i [simplified p]
  note step = prog-step [simplified p p′]
  interpret separated-tmps ts **by** fact
  have separated-tmps (ts[i := ((s′,t′), is @ is′, j,
    sb @ [Prog$_{\mathsf{sb}}$ (s,t) (s′,t′) is′], $\mathcal{D}$,$\mathcal{O}$)])
  proof (rule separated-tmps-nth-update [OF i-bound])
    from stmt-step-load-tmps-range [OF step] load-tmps-le [OF i-bound ts-i]
    stmt-step-tmps-count-mono [OF step]
    show $\forall$i$\in$load-tmps (is @ is′). i < t′
      by (auto simp add: load-tmps-append)
  next
    from read-tmps-le [OF i-bound ts-i] stmt-step-tmps-count-mono [OF step]
    show $\forall$i$\in$read-tmps (sb @ [Prog$_{\mathsf{sb}}$ (s, t) (s′, t′) is′]). i < t′
      by (auto simp add: read-tmps-append)
  next
    from stmt-step-data-store-sops-range [OF step] stmt-step-tmps-count-mono [OF step]
    store-sops-le [OF i-bound ts-i] valid-sops-stmt [OF i-bound ts-i]
    show $\forall$i$\in\bigcup$(fst ‘ store-sops (is @ is′)). i < t′
      by (fastforce simp add: store-sops-append)
  next
    from
      stmt-step-tmps-count-mono [OF step] write-sops-le [OF i-bound ts-i]
    show $\forall$i$\in\bigcup$(fst ‘ write-sops (sb @ [Prog$_{\mathsf{sb}}$ (s, t) (s′, t′) is′])). i < t′
      by (fastforce simp add: write-sops-append)
  next
    from tmps-le [OF i-bound ts-i]
    have dom j $\cup$ load-tmps is = {i. i < t} **by** simp
    with stmt-step-load-tmps-range′ [OF step] stmt-step-tmps-count-mono [OF step]
    show dom j $\cup$ load-tmps (is @ is′) = {i. i < t′}
      apply (clarsimp simp add: load-tmps-append)
      apply rule
      apply (drule union-eq-subsetD)
      apply fastforce

apply clarsimp
subgoal for x
apply (case-tac t ≤ x)
apply simp
apply (subgoal-tac x < t)
apply fastforce
apply fastforce
done
done
next
from valid-sops-stmt-invariant [OF prog-step [simplified p p′] valid-sops-stmt [OF i-bound ts-i]]
show valid-sops-stmt t′ s′.
next
show ∀(s′, t′)∈prog-configs (sb @ [$\text{Prog}_{\mathsf{sb}}$ (s, t) (s′, t′) is′]).
valid-sops-stmt t′ s′
proof −
{
fix s_1 t_1
assume cfgs: (s_1,t_1) ∈ prog-configs (sb @ [$\text{Prog}_{\mathsf{sb}}$ (s, t) (s′, t′) is′])
have valid-sops-stmt t_1 s_1
proof −
from valid-sops-stmt [OF i-bound ts-i]
have valid-sops-stmt t s.
moreover
from valid-sops-stmt-invariant [OF prog-step [simplified p p′] valid-sops-stmt [OF i-bound ts-i]]
have valid-sops-stmt t′ s′.
moreover
note valid-sops-stmt-sb [OF i-bound ts-i]
ultimately
show ?thesis
using cfgs
by (auto simp add: prog-configs-append)
qed
}
thus ?thesis
by auto
qed
qed
then
show ?thesis
by (simp add: p p′)
qed

lemma flush-step-sb-subset:
assumes step: (m,sb,𝒪) $\rightarrow_{\mathsf{f}}$ (m′, sb′,𝒪′)
shows set sb′ ⊆ set sb
using step
apply (induct c1==(m,sb,𝒪) c2==(m′,sb′,𝒪′) arbitrary: m sb 𝒪 acq m′ sb′ 𝒪′ acq

rule: flush-step.induct)
apply auto
done

lemma flush-step-preserves-separated-tmps:
  assumes i-bound: i < length ts
  assumes ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  assumes flush-step: (m,sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}$ (m', sb',$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$)
  assumes sep: separated-tmps ts
  shows separated-tmps (ts [i:=(p,is,j,sb',$\mathcal{D}$,$\mathcal{O}'$,$\mathcal{R}'$)])
proof −
  obtain s t **where** p: p=(s,t) **by** (cases p)
  note ts-i = ts-i [simplified p]
  interpret separated-tmps ts **by** fact
  have separated-tmps (ts [i:=((s,t),is,j,sb',$\mathcal{D}$,$\mathcal{O}'$,$\mathcal{R}'$)])
  proof (rule separated-tmps-nth-update [OF i-bound])
    from load-tmps-le [OF i-bound ts-i]
    show $\forall$i$\in$load-tmps is. i < t**.**
  next
    from flush-step-preserves-read-tmps [OF flush-step read-tmps-le [OF i-bound ts-i]]
    show $\forall$i$\in$read-tmps sb'. i < t**.**
  next
    from store-sops-le [OF i-bound ts-i]
    show $\forall$i$\in\bigcup$(fst ‘ store-sops is). i < t**.**
  next
    from flush-step-preserves-write-sops [OF flush-step write-sops-le [OF i-bound ts-i]]
    show $\forall$i$\in\bigcup$(fst ‘ write-sops sb'). i < t**.**
  next
    from tmps-le [OF i-bound ts-i]
    show dom j $\cup$ load-tmps is = {i. i < t}
      by auto
  next
    from valid-sops-stmt [OF i-bound ts-i]
    show valid-sops-stmt t s**.**
  next
    from valid-sops-stmt-sb [OF i-bound ts-i] flush-step-sb-subset [OF flush-step]
    show $\forall$(s', t')$\in$prog-configs sb'. valid-sops-stmt t' s'
      by (auto dest!: prog-configs-mono)
  qed
  then
  show ?thesis
    by (simp add: p)
qed

lemma sbh-step-preserves-store-sops-bound:
  assumes step: (is,j,sb,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$ (is',j',sb',m',$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$)
  assumes store-sops-le: $\forall$i$\in\bigcup$(fst ‘ store-sops is). i < t
  shows $\forall$i$\in\bigcup$(fst ‘ store-sops is'). i < t
  using step store-sops-le
  by cases auto

lemma sbh-step-preserves-write-sops-bound:
  assumes step: (is,j,sb,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$ (is′,j′,sb′,m′,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$)
  assumes store-sops-le: ∀ i∈⋃ (fst ‘ store-sops is). i < t
  assumes write-sops-le: ∀ i∈⋃ (fst ‘ write-sops sb). i < t
  shows ∀ i∈⋃ (fst ‘ write-sops sb′). i < t
  using step store-sops-le write-sops-le
  by cases (auto simp add: write-sops-append)

lemma sbh-step-prog-configs-eq:
  assumes step: (is,j,sb,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$ (is′,j′,sb′,m′,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$)
  shows prog-configs sb′ = prog-configs sb
using step
apply (cases)
apply (auto simp add: prog-configs-append)
done

lemma sbh-step-preserves-tmps-bound′:
  assumes step: (is,j,sb,m,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$ (is′,j′,sb′,m′,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$)
  shows dom j ∪ load-tmps is = dom j′ ∪ load-tmps is′
  using step
  apply cases
  apply (auto simp add: read-tmps-append)
  done

lemma sbh-step-preserves-separated-tmps:
  assumes i-bound: i < length ts
  assumes ts-i: ts!i = (p,is,j,sb,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
  assumes memop-step: (is, j, sb, m,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$
      (is′, j′, sb′, m′,$\mathcal{D}'$, $\mathcal{O}'$, $\mathcal{R}'$,$\mathcal{S}'$)
  assumes instr: separated-tmps ts
  shows separated-tmps (ts [i:=(p,is′,j′,sb′,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)])
proof −
  obtain s t **where** p: p=(s,t) **by** (cases p)
  note ts-i = ts-i [simplified p]
  interpret separated-tmps ts **by** fact
  have separated-tmps (ts [i:=((s,t),is′,j′,sb′,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)])
  proof (rule separated-tmps-nth-update [OF i-bound])
    from sbh-step-preserves-load-tmps-bound [OF memop-step load-tmps-le [OF i-bound ts-i]]
    show ∀ i∈load-tmps is′. i < t**.**
  next
    from sbh-step-preserves-read-tmps-bound [OF memop-step load-tmps-le [OF i-bound ts-i]
      read-tmps-le [OF i-bound ts-i]]
    show ∀ i∈read-tmps sb′. i < t**.**
  next
    from sbh-step-preserves-store-sops-bound [OF memop-step store-sops-le [OF i-bound ts-i]]
    show ∀ i∈⋃ (fst ‘ store-sops is′). i < t**.**

next
from sbh-step-preserves-write-sops-bound [OF memop-step store-sops-le [OF i-bound ts-i]
write-sops-le [OF i-bound ts-i]]
show $\forall$ i$\in\bigcup$(fst ‘ write-sops sb′). i < t**.**
next
from sbh-step-preserves-tmps-bound′ [OF memop-step] tmps-le [OF i-bound ts-i]
show dom j′ ∪ load-tmps is′ = {i. i < t}
by auto
next
from valid-sops-stmt [OF i-bound ts-i]
show valid-sops-stmt t s**.**
next
from valid-sops-stmt-sb [OF i-bound ts-i] sbh-step-prog-configs-eq [OF memop-step]
show $\forall$(s′, t′)∈prog-configs sb′. valid-sops-stmt t′ s′
by auto
qed
then show ?thesis
by (simp add: p)
qed

definition
valid-pimp ts ≡ separated-tmps ts

lemma prog-step-preserves-valid:
assumes i-bound: i < length ts
assumes ts-i: ts!i = (p,is,j,sb::stmt-config store-buffer,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
assumes prog-step: j⊢ p $\rightarrow_{\mathsf{s}}$ (p′, is′)
assumes valid: valid-pimp ts
shows valid-pimp (ts [i:=(p′,is@is′,j,sb@[Prog$_{\mathsf{sb}}$ p p′ is′],$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)])
using prog-step-preserves-separated-tmps [OF i-bound ts-i prog-step] valid
by (auto simp add: valid-pimp-def)

lemma flush-step-preserves-valid:
assumes i-bound: i < length ts
assumes ts-i: ts!i = (p,is,j,sb::stmt-config store-buffer,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
assumes flush-step: (m,sb,$\mathcal{O}$,$\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}$ (m′, sb′,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$)
assumes valid: valid-pimp ts
shows valid-pimp (ts [i:=(p,is,j,sb′,$\mathcal{D}$,$\mathcal{O}'$,$\mathcal{R}'$)])
using flush-step-preserves-separated-tmps [OF i-bound ts-i flush-step] valid
by (auto simp add: valid-pimp-def)

lemma sbh-step-preserves-valid:
assumes i-bound: i < length ts
assumes ts-i: ts!i = (p,is,j,sb::stmt-config store-buffer,$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)
assumes memop-step: (is, j, sb, m,$\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$
(is′, j′, sb′, m′,$\mathcal{D}'$, $\mathcal{O}'$, $\mathcal{R}'$, $\mathcal{S}'$)
assumes valid: valid-pimp ts
shows valid-pimp (ts [i:=(p,is′,j′,sb′,$\mathcal{D}'$,$\mathcal{O}'$,$\mathcal{R}'$)])
using

sbh-step-preserves-separated-tmps [OF i-bound ts-i memop-step] valid
by (auto simp add: valid-pimp-def)

lemma hd-prog-prog-configs: hd-prog p sb = p ∨ hd-prog p sb ∈ prog-configs sb
  by (induct sb) (auto split:memref.splits)

interpretation PIMP: xvalid-program-progress stmt-step λ(s,t). valid-sops-stmt t s valid-pimp
proof
  fix j p p′ is′
  assume step: j⊢ p $\rightarrow_{\mathsf{s}}$ (p′, is′)
  obtain s t **where** p: p = (s,t)
    by (cases p)
  obtain s′ t′ **where** p′: p′ = (s′,t′)
    by (cases p′)
  from prog-step-progress [OF step [simplified p p′]]
  show p′ ≠ p ∨ is′ ≠ []
    by (simp add: p p′)
next
  fix j p p′ is′
  assume step: j⊢ p $\rightarrow_{\mathsf{s}}$ (p′, is′)
    and valid-stmt: (λ(s, t). valid-sops-stmt t s) p
  obtain s t **where** p: p = (s,t)
    by (cases p)
  obtain s′ t′ **where** p′: p′ = (s′,t′)
    by (cases p′)
   from valid-sops-stmt-invariant [OF step [simplified p p′] valid-stmt [simplified p, simplified]]
  have valid-sops-stmt t′ s′**.**
  then show (λ(s, t). valid-sops-stmt t s) p′ **by** (simp add: p′)
next
  fix i ts p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
  assume i-bound: i < length ts
    and ts-i: ts ! i = (p, is, j, sb::(stmt × nat) memref list, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)
    and valid: valid-pimp ts
  from valid **have** separated-tmps ts
    by (simp add: valid-pimp-def)
  then interpret separated-tmps ts **.**
  obtain s t **where** p: p = (s,t)
    by (cases p)
  from valid-sops-stmt [OF i-bound ts-i [simplified p]]
  show (λ(s, t). valid-sops-stmt t s) p
    by (auto simp add: p)
next
  fix i ts p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb
  assume i-bound: i < length ts
    and ts-i: ts ! i = (p, is, j, sb::(stmt × nat) memref list, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)
    and valid: valid-pimp ts
  from valid **have** separated-tmps ts

by (simp add: valid-pimp-def)
then interpret separated-tmps ts **.**
obtain s t **where** p: p = (s,t)
by (cases p)
from hd-prog-prog-configs [of p sb] valid-sops-stmt [OF i-bound ts-i [simplified p]]
valid-sops-stmt-sb [OF i-bound ts-i [simplified p]]
show (λ(s, t). valid-sops-stmt t s) (hd-prog p sb)
by (auto simp add: p)
next
fix i ts p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p′ is′
assume i-bound: i < length ts
and ts-i: ts ! i = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)
and step: j⊢ p $\rightarrow_{\mathsf{s}}$ (p′, is′)
and valid: valid-pimp ts
show distinct-load-tmps is′ ∧
load-tmps is′ ∩ load-tmps is = {} ∧
load-tmps is′ ∩ read-tmps sb = {}
proof −
obtain s t **where** p: p=(s,t) **by** (cases p)
obtain s′ t′ **where** p′: p′=(s′,t′) **by** (cases p′)
note ts-i = ts-i [simplified p]
note step = step [simplified p p′]
from valid
interpret separated-tmps ts
by (simp add: valid-pimp-def)

from sbh-step-distinct-load-tmps-prog-step [OF step load-tmps-le [OF i-bound ts-i]
read-tmps-le [OF i-bound ts-i]]
show ?thesis **.**
qed
next
fix i ts p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p′ is′
assume i-bound: i < length ts
and ts-i: ts ! i = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)
and step: j⊢ p $\rightarrow_{\mathsf{s}}$ (p′, is′)
and valid: valid-pimp ts
show data-dependency-consistent-instrs (dom j ∪ load-tmps is) is′ ∧
load-tmps is′ ∩ $\bigcup$(fst ‘ store-sops is) = {} ∧
load-tmps is′ ∩ $\bigcup$(fst ‘ write-sops sb) = {}
proof −
obtain s t **where** p: p=(s,t) **by** (cases p)
obtain s′ t′ **where** p′: p′=(s′,t′) **by** (cases p′)
note ts-i = ts-i [simplified p]
note step = step [simplified p p′]
from valid
interpret separated-tmps ts
by (simp add: valid-pimp-def)

from sbh-valid-data-dependency-prog-step [OF step store-sops-le [OF i-bound ts-i]

write-sops-le [OF i-bound ts-i] valid-sops-stmt [OF i-bound ts-i]] tmps-le [OF i-bound ts-i]
    show ?thesis **by** auto
  qed
next
  fix i ts p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p′ is′
  assume i-bound: i < length ts
    and ts-i: ts ! i = (p, is, j, sb, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)
    and step: j⊢ p $\rightarrow_{\mathsf{s}}$ (p′, is′)
    and valid: valid-pimp ts
  show load-tmps is′ ∩ dom j = {}
  proof −
    obtain s t **where** p: p=(s,t) **by** (cases p)
    obtain s′ t′ **where** p′: p′=(s′,t′) **by** (cases p′)
    note ts-i = ts-i [simplified p]
    note step = step [simplified p p′]
    from valid
    interpret separated-tmps ts
      by (simp add: valid-pimp-def)
    from sbh-load-tmps-fresh-prog-step [OF step tmps-le′ [OF i-bound ts-i]]
    show ?thesis **.**
  qed
next
  fix j p p′ is
  assume step: j⊢ p $\rightarrow_{\mathsf{s}}$ (p′, is)
    and valid: (λ(s, t). valid-sops-stmt t s) p
  show ∀sop∈store-sops is. valid-sop sop
  proof −
    obtain s t **where** p: p=(s,t) **by** (cases p)
    obtain s′ t′ **where** p′: p′=(s′,t′) **by** (cases p′)
    note step = step [simplified p p′]
    from sbh-valid-sops-prog-step [OF step valid [simplified p,simplified]]
    show ?thesis **.**
  qed
next
  fix i ts p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb p′ is′
  assume i-bound: i < length ts
    and ts-i: ts ! i = (p, is, j, sb::stmt-config store-buffer, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)
    and step: j⊢ p $\rightarrow_{\mathsf{s}}$ (p′, is′)
    and valid: valid-pimp ts
  from prog-step-preserves-valid [OF i-bound ts-i step valid]
  show valid-pimp (ts[i := (p′, is @ is′, j, sb @ [Prog$_{\mathsf{sb}}$ p p′ is′], $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)]) **.**
next
  fix i ts p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb $\mathcal{S}$ m m′ sb′ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{S}'$
  assume i-bound: i < length ts
    and ts-i: ts ! i = (p, is, j, sb::stmt-config store-buffer, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)
    and step: (m, sb, $\mathcal{O}$, $\mathcal{R}$,$\mathcal{S}$) $\rightarrow_{\mathsf{f}}$ (m′, sb′,$\mathcal{O}'$,$\mathcal{R}'$,$\mathcal{S}'$)
    and valid: valid-pimp ts
  thm flush-step-preserves-valid [OF]
  from flush-step-preserves-valid [OF i-bound ts-i step valid]

show valid-pimp (ts[i := (p, is, j, sb′, $\mathcal{D}$, $\mathcal{O}'$,$\mathcal{R}'$)]) .
next
fix i ts p is $\mathcal{O}$ $\mathcal{R}$ $\mathcal{D}$ j sb $\mathcal{S}$ m is′ $\mathcal{O}'$ $\mathcal{R}'$ $\mathcal{D}'$ j′ sb′ $\mathcal{S}'$ m′
assume i-bound: i < length ts
and ts-i: ts ! i = (p, is, j, sb::stmt-config store-buffer, $\mathcal{D}$, $\mathcal{O}$,$\mathcal{R}$)
and step: (is, j, sb, m, $\mathcal{D}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{S}$) $\rightarrow_{\mathsf{sbh}}$
(is′, j′, sb′, m′,$\mathcal{D}'$, $\mathcal{O}'$, $\mathcal{R}'$,$\mathcal{S}'$)
and valid: valid-pimp ts
from sbh-step-preserves-valid [OF i-bound ts-i step valid]
show valid-pimp (ts[i := (p, is′, j′, sb′, $\mathcal{D}'$, $\mathcal{O}'$,$\mathcal{R}'$)]) .
qed

thm PIMP.concurrent-direct-steps-simulates-store-buffer-history-step
thm PIMP.concurrent-direct-steps-simulates-store-buffer-history-steps
thm PIMP.concurrent-direct-steps-simulates-store-buffer-step
We can instantiate PIMP with the various memory models.**interpretation** direct:
computation direct-memop-step empty-storebuffer-step stmt-step λp p′ is sb. ().
interpretation virtual:
computation virtual-memop-step empty-storebuffer-step stmt-step λp p′ is sb. ().
interpretation store-buffer:
computation sb-memop-step store-buffer-step stmt-step λp p′ is sb. sb .
interpretation store-buffer-history:
computation sbh-memop-step flush-step stmt-step λp p′ is sb. sb @ [$\text{Prog}_{\mathsf{sb}}$ p p′ is].

abbreviation direct-pimp-step::
(stmt-config,unit,bool,owns,rels,shared) global-config ⇒ (stmt-config,unit,bool,owns,rels,shared) global-config ⇒ bool
(‹- $\Rightarrow_{\mathsf{dp}}$ -› [60,60] 100)
where
c $\Rightarrow_{\mathsf{dp}}$ d ≡ direct.concurrent-step c d

abbreviation direct-pimp-steps::
(stmt-config,unit,bool,owns,rels,shared) global-config ⇒ (stmt-config,unit,bool,owns,rels,shared) global-config ⇒ bool
(‹- $\Rightarrow_{\mathsf{dp}}^*$ -› [60,60] 100)
where
direct-pimp-steps == direct-pimp-step^**
Execution examples**lemma** Assign-Const-ex:
([((Assign True (Tmp ({},λj. a)) (Const c) (λj. A) (λj. L) (λj. R) (λj. W),t),[],j,(),$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)],m,$\mathcal{S}$) $\Rightarrow_{\mathsf{dp}}^*$
([((Skip,t),[],j,(),True,$\mathcal{O}$ ∪ A − R,Map.empty)],m(a := c),$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
apply (rule converse-rtranclp-into-rtranclp)
apply (rule direct.Program [**where** i=0])
apply simp
apply simp
apply (rule Assign)
apply simp
apply (rule converse-rtranclp-into-rtranclp)
apply (rule direct.Memop [**where** i=0])
apply simp

apply simp
apply (rule direct-memop-step.WriteVolatile)
apply simp
done

lemma
([((Assign True (Tmp ({},λj. a)) (Binop (+) (Mem True x) (Mem True y)) (λj. A) (λj. L) (λj. R) (λj. W),t),[],j,(),$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)],m,S)
$\Rightarrow_{\mathsf{dp}}{}^*$
([((Skip,t + 2),[],j(t$\mapsto$m x, t + 1 $\mapsto$m y),(),True,$\mathcal{O}$ $\cup$ A $-$ R,Map.empty)],m(a := m x + m y),S $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
apply (rule converse-rtranclp-into-rtranclp)
apply (rule direct.Program [**where** i=0])
apply simp
apply simp
apply (rule Assign)
apply simp

apply (rule converse-rtranclp-into-rtranclp)
apply (rule direct.Memop)
apply simp
apply simp
apply (rule direct-memop-step.Read)
apply simp

apply (rule converse-rtranclp-into-rtranclp)
apply (rule direct.Memop)
apply simp
apply simp
apply (rule direct-memop-step.Read)
apply simp

apply (rule converse-rtranclp-into-rtranclp)
apply (rule direct.Memop)
apply simp
apply simp
apply (rule direct-memop-step.WriteVolatile)
apply simp
done

lemma
assumes isTrue: isTrue c
shows
([((Cond (Const c) (Assign True (Tmp ({},λj. a)) (Const c) (λj. A) (λj. L) (λj. R) (λj. W)) Skip,t) ,[],j,(),$\mathcal{D}$,$\mathcal{O}$,$\mathcal{R}$)],m,$\mathcal{S}$) $\Rightarrow_{\mathsf{dp}}{}^*$
([((Skip,t),[],j,(),True,$\mathcal{O}$ $\cup$ A $-$ R,Map.empty)],m(a := c),$\mathcal{S}$ $\oplus_{\mathsf{W}}$ R $\ominus_{\mathsf{A}}$ L)
apply (rule converse-rtranclp-into-rtranclp)
apply (rule direct.Program [**where** i=0])
apply simp

apply simp
apply (rule Cond)
apply simp
apply simp

apply (rule converse-rtranclp-into-rtranclp)
apply (rule direct.Program [**where** i=0])
apply simp
apply simp
apply (rule CondTrue)
apply simp
apply (simp add: isTrue)
apply simp

apply (rule Assign-Const-ex)
done

end

References

1. Advanced Micro Devices (AMD), Inc. *AMD64 Architecture Programmer's Manual: Volumes 1–3*. September 2007. rev. 3.14.
2. Sarita V. Adve and Kourosh Gharachorloo. Shared memory consistency models: A tutorial. *IEEE Computer*, 29(12):66–76, 1996.
3. David Aspinall and Jaroslav Sevcík. Formalising Java's data race free guarantee. In Klaus Schneider and Jens Brandt, editors, *TPHOLs*, volume 4732, pages 22–37, 2007.
4. Clemens Ballarin. Locales and locale expressions in Isabelle/Isar. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *Types for Proofs and Programs, International Workshop, TYPES 2003, Torino, Italy, April 30 – May 4, 2003, Revised Selected Papers*, volume 3085, pages 34–50. Springer, 2003.
5. Clemens Ballarin. Interpretation of locales in Isabelle: Theories and proof contexts. In Jonathan M. Borwein and William M. Farmer, editors, *Mathematical Knowledge Management, 5th International Conference, MKM 2006, Wokingham, UK, August 11–12, 2006, Proceedings*, volume 4108, pages 31–43. Springer, 2006.
6. Sebastian Burckhardt and Madanlal Musuvathi. Effective program verification for relaxed memory models. In *CAV '08: Proceedings of the 20th international conference on Computer Aided Verification*, pages 107–120, Berlin, Heidelberg, 2008. Springer-Verlag.
7. Geng Chen, Ernie Cohen, and Mikhail Kovalev. Store buffer reduction with MMUs. In Dimitra Giannakopoulou and Daniel Kroening, editors, *Verified Software: Theories, Tools and Experiments*, pages 117–132, Cham, 2014. Springer International Publishing.
8. Ernie Cohen and Bert Schirmer. From total store order to sequential consistency: A practical reduction theorem. In Matt Kaufmann, Lawrence Paulson, and Michael Norrish, editors, *Interactive Theorem Proving (ITP 2010)*, volume 6172 of *Lecture Notes in Computer Science*, Edinburgh, UK, 2010. Springer.
9. Intel. Intel 64 architecture memory ordering white paper. SKU 318147-001, 2007.
10. Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual: Volumes 1–3b*. 2009. rev. 29.
11. Tobias Nipkow. Winskel is (almost) right: Towards a mechanized semantics textbook. In V. Chandru and V. Vinay, editors, *Foundations of Software Technology and Theoretical Computer Science*, volume 1180, pages 180–192, 1996.
12. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283. Springer, 2002.
13. Jonas Oberhauser. A simpler reduction theorem for x86-tso. In Arie Gurfinkel and Sanjit A. Seshia, editors, *Verified Software: Theories, Tools, and Experiments*, pages 142–164, Cham, 2016. Springer International Publishing.

14. Scott Owens. Reasoning about the implementation of concurrency abstractions on x86-TSO. Technical report, University of Cambridge, 2009.
15. Scott Owens, Susmit Sarkar, and Peter Sewell. A better x86 memory model: x86-TSO. In *22nd International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2009)*, 2009.
16. Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828. Springer, 1994.
17. Tom Ridge. Operational reasoning for concurrent Caml programs and weak memory models. In Klaus Schneider and Jens Brandt, editors, *Theorem Proving in Higher Order Logics, 20th International Conference, TPHOLs 2007, Kaiserslautern, Germany, September 10-13, 2007, Proceedings*, volume 4732, pages 278–293, 2007.
18. Jaroslav Sevcík and David Aspinall. On validity of program transformations in the Java memory model. In Jan Vitek, editor, *ECOOP*, volume 5142, pages 27–51, 2008.