Visibility Reasoning for Concurrent Snapshot Algorithms

JOAKIM ÖHMAN, IMDEA Software Institute, Spain and Universidad Politécnica de Madrid, Spain
ALEKSANDAR NANEVSKI, IMDEA Software Institute, Spain

Visibility relations have been proposed by Henzinger et al. as an abstraction for proving linearizability of concurrent algorithms that obtains modular and reusable proofs. This is in contrast to the customary approach based on exhibiting the algorithm's linearization points. In this paper we apply visibility relations to develop modular proofs for three elegant concurrent snapshot algorithms of Jayanti. The proofs are divided by signatures into components of increasing level of abstraction; the components at higher abstraction levels are shared, i.e., they apply to all three algorithms simultaneously. Importantly, the interface properties mathematically capture Jayanti's original intuitions that have previously been given only informally.

CCS Concepts: • **Theory of computation → Concurrent algorithms**; **Program verification**; *Program specifications*; • **Software and its engineering** → *Formal software verification.*

Additional Key Words and Phrases: concurrent snapshots, visibility relations, linearizability

ACM Reference Format:
Joakim Öhman and Aleksandar Nanevski. 2022. Visibility Reasoning for Concurrent Snapshot Algorithms. *Proc. ACM Program. Lang.* 6, POPL, Article 33 (January 2022), 30 pages. https://doi.org/10.1145/3498694

1 INTRODUCTION

Linearizability [Herlihy and Wing 1990] is a standard correctness condition for concurrent data structures. It requires that the operations in any execution over the data structure may be ordered sequentially, without incurring changes to the observed results. In other words, the methods of a linearizable concurrent structure exhibit the same external behavior as the sequential equivalent. Programmers can use the concurrent variant to efficiently utilize modern systems' multi-core setup, and rely on the sequential variant for understanding and formal reasoning.

Many methods exist for proving linearizability of a data structure. The standard idea shared by most of them involves finding, for each method of the structure, a point in time in the concurrent execution when the method can be considered as *logically* occurring. In other words, a method may execute over a period of time, admitting interference from other concurrent threads; nevertheless, for all reasoning intents and purposes, the execution is indistinguishable from one where the method executes atomically at a single point in time, without any interference. This point in time is referred to as the *linearization point*. Where the linearization point lies for a given method may depend on the run-time behavior and interleaving of other methods executing concurrently, sometimes even including behavior that occurs after the original method has already terminated (the latter are often termed *far future* linearization points). Although significant progress has been made recently in the verification of concurrent structures and algorithms, and in particular by the introduction of so-called *prophecy variables* [Abadi and Lamport 1991; Jacobs et al. 2018; Jung et al. 2020; Lynch and

Authors' addresses: Joakim Öhman, joakim.ohman@imdea.org, IMDEA Software Institute, Madrid, Spain and Universidad Politécnica de Madrid, Pozuelo de Alarcón, Spain; Aleksandar Nanevski, aleks.nanevski@imdea.org, IMDEA Software Institute, Madrid, Spain.

2475-1421/2022/1-ART33
https://doi.org/10.1145/3498694

Vaandrager 1995] to model the dependence of linearization points on run-time behavior (including in the far future), establishing linearizability by explicitly exhibiting the linearization points of an algorithm remains a highly complex task in general.

A different approach, which avoids explicit reasoning about linearization points, has been proposed by Henzinger et al. [2013]. In this approach, one first specifies a set of properties and proves that the properties hold of every execution history over the data structure. One then constructs a proof of linearizability using only the specified properties as axioms, thereby abstracting from the underlying executions. In other words, the low-level reasoning about the concrete placement of linearization points is replaced by higher-level reasoning out of data structure axioms, in turn facilitating proof decomposition, modularity, abstraction, and reuse.

The properties used in the axiomatization are defined under relations tying an event in the execution to other events that depend on it, i.e., that *observe* it. For example, the event of reading of a pointer will be related to the write event that was responsible for mutating the pointer. Once the pointer is mutated by another write, the subsequent reads will observe (i.e., be related to) the new write, or possibly another later write. Similar ideas of reasoning about event observations have been used in the axiomatizations of weak memory models [Raad et al. 2019] and in distributed systems [Viotti and Vukolić 2016] where they have been captured by means of so-called *visibility* relations. Reasoning by visibility has also been applied in different ways to prove linearizability of concrete data structures, such as queues [Henzinger et al. 2013] and stacks [Dodds et al. 2015], including reasoning automation [Bouajjani et al. 2017], and reasoning about relaxed notions of linearizability [Emmi and Enea 2019; Krishna et al. 2020].

In this paper, we demonstrate the further applicability of reasoning by visibility, by applying it in a novel way to even more nuanced algorithms and proofs. In particular, we show how visibility can be used to express and axiomatize the important *internal* properties shared by several *snapshot* algorithms. Similarly to the approach of Henzinger et al. [2013] to queues, the axiomatization enables a modularization of the linearizability proof: a significant portion of the proof is carried out once, and then reused for each snapshot algorithm.

More specifically, we verify the three snapshot algorithms by Jayanti [2005]. A snapshot algorithm scans a memory array and returns the values read, so that the obtained values reflect the state of the array at one point in time. In a sequential setting this is trivial to achieve since the array remains unchanged during the scan. However, Jayanti's algorithms are concurrent, allowing interfering threads to modify the array while a scan is in progress. Jayanti's algorithms are of increasing efficiency and generality. The simplest is the single-writer/single-scanner algorithm, which assumes that no two scanners run concurrently, and that no two writers concurrently modify the same array element. Jayanti's second algorithm generalizes to a multi-writer/single-scanner setting, and the third is the most general and ultimately desirable multi-writer/multi-scanner version. Jayanti describes the linearization points only for the single-writer/single-scanner algorithm, but already this description is quite challenging to transform into a fully formal correctness proof [Delbianco et al. 2017] because the algorithm exhibits far future linearization points.

Of interest to us in the current paper is that each of the three algorithms builds on the previous one by relaxing some part of the previous algorithm's implementation, while preserving the essential invariants that Jayanti calls *forwarding principles*. Jayanti credits the forwarding principles as the key idea behind his design, because the principles are shared by the three algorithms, and abstractly govern how a concurrent write into the array should be "forwarded" to a scanner that is in progress, but has already read past the written element.

In this paper, we show how the forwarding principles can be axiomatized mathematically in terms of visibility, which we present in Section 2 (Jayanti states the principles in English, much less formally). In Section 3, we develop the linearizability proof out of the axioms alone, so that it

applies to all three algorithms simultaneously. We then establish that the axioms hold for each of the three algorithms: in Section 3 for the first algorithm and Section 4 for the second and third. In Section 5 we apply our method to verify another snapshot algorithm, that of Afek et al. [1993].

By employing visibility, we sidestep the difficulties inherent in reasoning about linearization points in general, and far future linearization points in particular. Our development substantiates that visibility is a natural abstraction to use in specifications and proofs, as it enables formally capturing the intuition—that of forwarding principles—that motivated the design of Jayanti's algorithms in the first place. In summary, our contributions are as follows.

- This is the first formal proof of all of Jayanti's three snapshot algorithms. Moreover, it efficiently reuses proofs between algorithms. Delbianco et al. [2017] employed a variation of the linearization point approach to prove only Jayanti's first algorithm, with no direct way of extending the proof to the other two algorithms, which are significantly more involved. That proof was mechanized in Coq. Based on *loc. cit.*, Jacobs [2018] developed a mechanized proof of Jayanti's first algorithm in VeriFast, using prophecy variables.
- We have axiomatized the forwarding principles, which were Jayanti's motivating insight, and key properties of his snapshot algorithms, but have so far been out of reach of formal mathematics. The axiomatization is non-trivial, and required generalizing from Jayanti's English description in order to apply to all three algorithms. It also enabled us to prove, for the first time, that forwarding principles formally imply linearizability. While visibility relations have been used before to axiomatize concurrent structures, this shows that they can also usefully capture important *internal* properties.

2 OVERVIEW

2.1 Jayanti's First Snapshot Algorithm

The n-snapshot data-structure is an array of length n, which consists of two kinds of operations: SCAN which reads the memory and returns a list of length n reflecting the state of the memory at a point in time; and WRITE(i, v) which writes the value v into memory cell i. We use s to range over instances of SCAN, w_i to range over instances of WRITE(i, v) for some v, and e to range over all operations in general. In line with standard terminology of linearizability, we call these operations *abstract events* or *abs events* and color them magenta.

A snapshot (or any other) algorithm is linearizable if for every concurrent execution, there exists some way of sequentially ordering the *overlapping* (abs) events so that the value returned by each of the events remains unchanged compared to the concurrent execution, and moreover, matches the intended semantics of the data structure. Intuitively, linearizability implies that we can re-run the computation sequentially to obtain the same results as in the original concurrent run; however, the internal state of the algorithm during and after the runs need not match.

Implementing an efficient and correct concurrent snapshot algorithm is more challenging than it may seem. To highlight this point, consider a naïve implementation, where writers simply write to a shared array `A` and scanners simply iterate over the array `A` to obtain a snapshot. To allow this implementation to operate efficiently, we allow writes and scans to run concurrently. For this simple implementation, consider a snapshot array of length 2, where we have a scan s running concurrently with writes w_0 writing 2 and w_1 writing 3. Let 0 be the initial value of each array cell, and consider the following execution:

- Scan s starts and reads 0 from `A[0]`, after which the scheduler interrupts s.
- Write w_0 starts, writing 2 to `A[0]`, and after w_0 finishes, write w_1 writes 3 to `A[1]`.
- Scan s resumes, reading 3 from `A[1]`. It then returns the snapshot $(0, 3)$.

```
resource A : array[n] of val
resource B : array[n] of val ∪ {⊥}
resource X : 𝔹 := false

1: WRITE(i : ℕ, v : val) ≜                          ▷ w_i
2:     A[i] := v                                    ▷ w_i a
3:     x ← X                                        ▷ w_i x
4:     if x then B[i] := v                          ▷ w_i b

5: SCAN: array[n] of val ≜                          ▷ s
6:     X := true                                    ▷ s_on
7:     for i ∈ {0 ... n − 1} do
8:         B[i] := ⊥                                ▷ sr_i
9:     for i ∈ {0 ... n − 1} do
10:        a ← A[i]                                 ▷ sa_i
11:        V[i] := a
12:    X := false                                   ▷ s_off
13:    for i ∈ {0 ... n − 1} do
14:        b ← B[i]                                 ▷ sb_i
15:        if b ≠ ⊥ then V[i] := b
16:    return V
```

Algorithm 1. Jayanti's single-writer, single-scanner snapshot algorithm over a memory of length n. Mnemonics on the right identify the corresponding commands.

The snapshot $(0, 3)$ should indicate that there exists a point in time when the array consisted of that pair, however that is not the case. The array started as $(0, 0)$, followed by $(2, 0)$ after write w_0, and $(2, 3)$ after write w_1, but none of these states are reflected in the result. In a sense, the scan s missed the write w_0, yet it caught the write w_1, which occurred after w_0. Jayanti's snapshot algorithms ensure that writes are not missed by the scanner, as we explain next.

Algorithm 1 is the first and simplest of Jayanti's snapshot algorithms. The idea is for a scan to make two passes over the memory, first over the main array A, and then over the auxiliary array B. A writer updates the array B if it detects a concurrent scan via the boolean flag X, to *forward* its value. That is, in case the scanner missed the writer's value when scanning A, it will have a chance to catch the value when scanning B. We thus refer to B as the *forwarding array*. Algorithm 1 is a single-writer/single-scanner algorithm, meaning that for it to behave correctly, two scans must not run concurrently and two writers must not concurrently mutate the same array cell. In an implementation, this can be enforced by explicit locking, which we elide from Algorithm 1 following Jayanti's original presentation.

Describing the algorithm in more detail, SCAN works by first setting X to `true` by event s_{on}, signaling that a scan is running the first pass. This is followed by clearing the forwarding array B by setting all its cells to $\bot$ via the sr_i events, for each i. The clearing ensures that the current scan cannot consider the forwards left over from the previous scans. Next, the scanner creates a naïve snapshot by copying the main array A by the events sa_i for each i into the local array V. However, as we argued before, this is insufficient for a correct snapshot. This is where the second pass comes in, which starts after X is set to `false` by event s_{off}. In the second pass, the procedure repairs the naïve snapshot by stepping through the forwarding array B by the events sb_i for each i. If a non-$\bot$ (i.e., forwarded) value is found, it overwrites the original value in V, thus repairing the snapshot and preventing missing writes. Finally, the scanner returns V, which contains the complete snapshot.

For WRITE(i, v), it starts by writing its value to A by event $w_i a$, followed by a check for a concurrently running scanner performing the naïve pass of the scan by event $w_i x$. If such a scan is detected, the writer forwards its value to B by event $w_i b$.

Events marked in blue are called *representation* or *rep* events, and are used internally in the implementation of abs (i.e., magenta) events. The distinction between abs and rep events is standard in the theory of linearizability [Herlihy and Wing 1990]. We will use the naming convention whereby abs events and rep events with the same priming belong together, e.g. we assume sa_i belongs to s and $w_i' a$ belongs to w_i'.

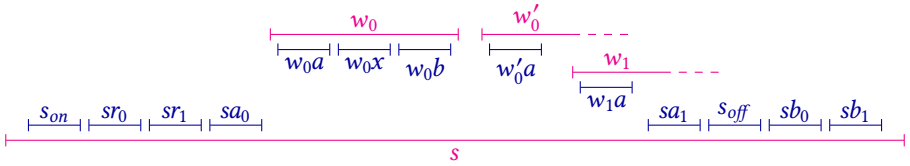


Fig. 1. Diagram illustrating an execution scenario for Algorithm 1 with an array length of two.

Now, consider the scenario illustrated in Fig. 1, where we have Algorithm 1 being executed over an array of length two, with each value initially set to 0, with write events w_0, w'_0 and w_1 writing 2, 3 and 4 respectively, and a scan s. When s reads A[0] with sa_0 for its first pass, it reads 0, while when it reads A[1] with sa_1 it reads 4 written by w_1 with w_1a. Between the two passes of s, w_0a is missed, however since X is set to `true` by s_{on}, w_0 will forward the value 2 with w_0b. At the second pass, s reads B[0] with sb_0, reading 2 written by w_0b, and when it reads B[1] with sb_1, it reads $\perp$ written by sr_1, meaning s will use the original value 4 for its final snapshot, thus returning (2,4).

In contrast to $(0, 3)$ before, linearizability admits $(2, 4)$ as a correct snapshot even though A never contained $(2, 4)$ during the execution. Indeed, A only contained $(0, 0)$, $(2, 0)$, $(3, 0)$ and $(3, 4)$. This is actually fine, because we can reorder the concurrent events with the order $w_0 \rightarrow w_1 \rightarrow s \rightarrow w'_0$, which, when executed sequentially, result in s having snapshot $(2, 4)$. In the physical execution, the event w_0 returned before w'_0 and w_1 started. The reordering respects this by listing w_0 before w'_0 and w_1. In other words, the reordering affects only events that physically overlapped, as required by linearizability.

Jayanti sketches the correctness proof of Algorithm 1 by describing its linearization points. The linearization point for a scan s is always when the scan performs s_{off}. However, the linearization point of a write w_i varies. If there is no scan concurrent to w_i, or there is a concurrent scan, but it reads the value of w_i either from A or from B, then the writer's linearization point is at w_ia. If there is a concurrent scan s that misses w_i, which can occur if the scanner misses w_i in its A pass, and w_i either does not write into B[i] due to X being set to `false` before the writer could forward, or w_i writing into B[i] too late, then w_i must be logically considered as occurring after s. Thus, the linearization point of w_i is immediately after s_{off}, making the linearization point of WRITE *external*, as its position is given in terms of another procedure, in this case SCAN. The observation that a scan missed a write w_i, which occurs when the scan reads B[i] with sb_i, can be made after the writer has already terminated, making WRITE exhibit a far future linearization point.

We proceed to show how to organize the linearizability proof of Algorithm 1, and the other two Jayanti algorithms, by axiomatizing forwarding via visibility, without using linearization points.

2.2 Basic Abstractions of Visibility Reasoning

2.2.1 Events and Their Structure. An event is an object consisting of fields start, end, op, in, and out, describing the following aspects of the execution of some operation of the data structure: start is the event's beginning time, end is the ending time, op is the operation name (e.g., SCAN or WRITE), in is the operation's input, and out is the output. We refer to the elements of an event e by projection, e.g. e.start and e.op. The fields e.start and e.end are natural numbers, and e.end may be ∞ (infinity) to represent that e has not terminated yet, i.e., e is terminated iff e.end $\neq \infty$, which we denote with $\mathcal{T}(e)$. For every e, e.end $>$ e.start. The types of e.in and e.out depend on e.op, and e.out is undefined iff e.end $= \infty$.

Additionally, each event e contains the optional field parent, corresponding to the event that invoked e, if any. For example, if e is a rep event, then e.parent is the abs event that contains e. Note that e.parent and e need not have the same op, in and out fields; e.g., a write rep event can be invoked both by abs writer and abs scanner.

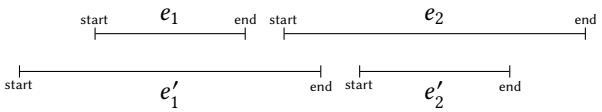


Fig. 2. Visual representation of events highlighting the interval and subevent Properties (RB.1) and (RB.2).

Finally, we require that each abs event is single-threaded, and thus cannot fork children threads.

We denote the set of all events of a given execution history by Events. If $E \subset$ Events, then $op(E) = \{e \in E \mid e.\text{op} = op\}$ selects the events with operation op, and we overload $\mathcal{T}$ over sets with $\mathcal{T}(E) = \{e \in E \mid \mathcal{T}(e)\}$ to select the terminated events.

As linearization order can only affect physically overlapping events, proving linearizability requires reasoning about non-overlapping events, which is captured by the *returns-before* relation

$$e \sqsubset e' \quad \triangleq \quad e.\text{end} < e'.\text{start}$$

denoting that e terminated before e' started. Two events are overlapping if they are unrelated by $\sqsubset$. As customary, we write $\sqsubseteq$ for the reflexive closure of $\sqsubset$. The relation $\sqsubset$ is irreflexive (i.e. *acyclic*) partial order, and moreover, an *interval* order [Felsner 1992], as it satisfies the following property

$$e_1 \sqsubset e_2 \wedge e_1' \sqsubset e_2' \implies e_1 \sqsubset e_2' \vee e_1' \sqsubset e_2 \tag{RB.1}$$

Fig. 2 illustrates why Property (RB.1) must hold. The figure shows events e_1, e_2, e_1', e_2' such that $e_1 \sqsubset e_2$, and $e_1' \sqsubset e_2'$, and Property (RB.1) holds because also $e_1 \sqsubset e_2'$. We can try to invalidate the latter by shifting e_1' and e_2' to the left so that $e_2'.\text{start} < e_1.\text{end}$ while maintaining $e_1'.\text{end} < e_2'.\text{start}$. But then we're forced to have $e_1'.\text{end} < e_2.\text{start}$, i.e. $e_1' \sqsubset e_2$ which re-establishes Property (RB.1).

We also say that e is a *subevent* of e' (alternatively, e' *contains* e) if

$$e \subseteq e' \quad \triangleq \quad e'.\text{start} \leq e.\text{start} \wedge e.\text{end} \leq e'.\text{end}$$

For any rep event e, we require that e.parent must be an abs event such that $e \subseteq e$.parent. Additionally, the following property holds for subevents and returns-before.

$$e_1 \subseteq e_1' \wedge e_2 \subseteq e_2' \wedge e_1' \sqsubset e_2' \implies e_1 \sqsubset e_2 \tag{RB.2}$$

That is, if e_1' returned before e_2' then all subevents of e_1' must return before any subevent of e_2'. For example, in Fig. 2, $e_1 \subseteq e_1'$ and $e_2' \subseteq e_2$, and $e_1 \sqsubset e_2'$. If we shift e_1' to the left so that $e_1' \sqsubset e_2$, then we just increase the distance between e_1 and e_2', maintaining $e_1 \sqsubset e_2'$.

2.2.2 Visibility Relations. If the event e' depends on the result of e, we say that e is *visible* to e', or alternatively that e is *observed* by e'. We denote the relationship as

$$e \prec e'$$

Depending on the data structure being verified, we will often require several different observation relations to differentiate how the observation came about. For example, in the case of Jayanti, we will use $\xrightarrow{\text{rf}}$ for a "reads-from observation" (a reader observes a writer by reading what was written), and $\xrightarrow{\text{fwd}}$ for a "forwarding observation" (a scanner observes a writer by having the written value forwarded). The former relation is tied to the physical act of reading a pointer, and will be the same in all three algorithms. The latter relation differs for different algorithms and is typically provided by the human verifier, similarly to how loop invariants must often be provided.

We will typically obtain the visibility relation $\prec$ by unioning all the different observation subrelations. We then require the following property of $\prec$

$$e \prec^+ e' \implies e' \not\sqsubseteq e \tag{V.1}$$

where $\prec^+$ is the transitive closure of $\prec$. Read contrapositively, the property says that if e' terminated before e started, then e' cannot end a non-empty sequence of observations starting from e. In particular, as a special case, e' cannot observe e or be equal to e, meaning $\prec$ has to be irreflexive. We write $\preccurlyeq$ and $\xrightarrow{\mathbf{rf}}{}^{=}$ for the reflexive closure of $\prec$ and $\xrightarrow{\mathbf{rf}}$ respectively.

2.2.3 Happens-Before Relation. Given two events e and e', if $e \sqsubset e'$ or $e \prec e'$, then clearly, in the ultimate linearization order we want to construct, e must appear before e'. To capture this intuition, we define the *happens-before* relation as the transitive closure

$$\lhd \;\triangleq\; (\sqsubset \cup \prec)^+$$

We also name the *single-step happens-before* relation $\lhd_1 = (\sqsubset \cup \prec)$, so that $\lhd = \lhd_1^* = \preccurlyeq\lhd_1 = \lhd_1\preccurlyeq = \lhd_1^+$, where $\preccurlyeq$ is the reflexive-transitive closure $\lhd_1^*$. The $\sqsubset$, $\prec$ and $\lhd$ relations are all standard in the literature [Viotti and Vukolić 2016]. An important property is that $\lhd$ is an irreflexive (i.e., *acyclic*) partial order, which is ensured by $\prec$ satisfying Property (V.1).

Lemma 2.1. *If visibility relation $\prec$ satisfies Property (V.1) then $\lhd$ is irreflexive.*

Proof. We assume that $\lhd$ is not irreflexive (i.e., there exists e such that $e \lhd e$), and derive contradiction. The relation $e \lhd e$ is a cyclic chain of $\lhd_1$, each of which is either $\sqsubset$ or $\prec$. We are justified in considering chains with only one or zero occurrences of $\sqsubset$, as chains with more occurrences of $\sqsubset$ can be shortened, thus we can recursively shorten a chain until it consists of one or zero $\sqsubset$. Indeed, if the chain has more than one $\sqsubset$, it has the form

$$e \lhd_1 \cdots \lhd_1 e_1 \sqsubset e_2 \prec \cdots \prec e_3 \sqsubset e_4 \lhd_1 \cdots \lhd_1 e\,,$$

where e_2 and e_3 are related by a chain of zero or more $\prec$'s (i.e., $e_2 = e_3$ or $e_2 \prec^+ e_3$). But then, we can remove one $\sqsubset$ as follows. If $e_2 = e_3$, by transitivity of $\sqsubset$, we can shorten the chain to $e \preccurlyeq e_1 \sqsubset e_4 \preccurlyeq e$. If $e_2 \prec^+ e_3$, by Property (RB.1), it must be either $e_1 \sqsubset e_4$ or $e_3 \sqsubset e_2$. If $e_1 \sqsubset e_4$, we again shorten to $e \preccurlyeq e_1 \sqsubset e_4 \preccurlyeq e$. Otherwise, $e_3 \sqsubset e_2$ and $e_2 \prec^+ e_3$ contradict Property (V.1).

On the other hand, if there is exactly one $\sqsubset$ in the chain, i.e., $e \prec^* e_1 \sqsubset e_2 \prec^* e$, then $e_2 \prec^* e \prec^* e_1$, i.e., $e_2 \prec^* e_1$ which, along with $e_1 \sqsubset e_2$, contradicts Property (V.1). Finally, if there is no $\sqsubset$ in the chain, i.e., $e \prec^+ e$, then Property (V.1) directly implies the contradiction $e \not\sqsubseteq e$. □

We will usually color the relations with the same colors as the events they are relating (e.g., $e \prec e'$ for abs events e and e' and $e_r \lhd e_r'$ for rep events e_r and e_r').

2.2.4 Memory Model. With the above relations we can now state as axioms the properties that we expect of the underlying memory model. We start with a simple axiomatization of memory that is sufficient for Algorithm 1. We will extend this axiomatization in Section 4 to account for the **LL** (load-link) and **SC** (store-conditional) memory operations, required by Algorithms 2 and 3.

The memory model only considers operations over individual memory cells, aka. atomic registers [Herlihy and Shavit 2008]. Their axiomatization in terms of visibility relations is given in Fig. 3, in the form of a signature we assume the memory to satisfy. We use blue in this figure, since we will only use events of atomic registers as rep events. The sets W and R are defined by the history as the sets of writes and reads respectively. The quantifier Σ signifies that the relations $\prec$ and $\xrightarrow{\mathbf{rf}}$ are abstract components of the specification; the clients do not know anything about these components outside of the listed axioms. Given an instance X of the signature, the clients can refer to the relations and to the let definitions by projection as $X.\prec$, $X.\xrightarrow{\mathbf{rf}}$, $X.W$ and $X.R$. We will present similar signatures for snapshot data-structures and Jayanti-style forwarding.

Looking at Fig. 3, an atomic register is mathematically represented by two sets of events: W, corresponding to all the writes into the register's memory cell, and R, corresponding to all the

signature AReg $\triangleq$		**Atomic Register**
Let	W	Set of all writes of the register of the history
	R	Set of all reads of the register of the history
$\sum$	$\prec \subseteq (W \cup R)^2$	Visibility relation
	$\xrightarrow{\text{rf}} \subseteq \prec \cap W \times R$	Reads-from visibility
Let	$\lhd \triangleq (\sqsubset \cup \prec)^+$	Happens-before order
$\forall e, e'.$	$e \prec^+ e' \implies e' \not\sqsubset e$	(V.1)
$\forall r \in \mathcal{T}(R).$	$\exists w \in W.\ w \xrightarrow{\text{rf}} r \wedge w.\text{in} = r.\text{out}$	(M.1)
$\forall w \in W, r \in R.$	$w \xrightarrow{\text{rf}} r \implies \nexists w'.\ w \lhd w' \lhd r$	(M.2)
$\forall w, w' \in W, r \in R.$	$w \xrightarrow{\text{rf}} r \wedge w' \xrightarrow{\text{rf}} r \implies w = w'$	(M.3)
$\forall w, w' \in W.$	$w \neq w' \implies w \lhd w' \vee w' \lhd w$	(M.4)

Fig. 3. Signature representation of the properties of an atomic register.

reads. These are related by $w \xrightarrow{\text{rf}} r$, stating that read r *reads from* (i.e., observes) write w. These events may also be related by some other relation part of the total set of observations $\prec$, which includes $\xrightarrow{\text{rf}}$; Property (V.1) must hold of $\prec$ to ensure that $\lhd$ is irreflexive (thus, a partial order) as per Lemma 2.1. Property (M.1) states that for any terminated read r there exists an observed write w with its input being the same as the output of r. Property (M.2) states that the read r can observe only the latest write w into the given memory cell. Property (M.3) states that a read may observe at most one write. Property (M.4) states that the writes into the given cell are totally ordered. These are standard properties of sequentially consistent memory [Herlihy and Shavit 2008].

We want to reason about $\lhd$ order arbitrarily between rep events, even if they belong to distinct register objects. As an analogue to the locality property of linearizability, which says that the union of multiple linearizable objects is itself linearizable, we establish that the union of objects satisfying Property (V.1) itself satisfies Property (V.1). By Lemma 2.1, this implies that $\lhd$ order between any rep event is a partial order. More formally, let $R_1 \ldots R_m$ be all objects satisfying AReg and representing our memory. We define the top-level $\prec$ (and top-level $\lhd$) relation over rep events as the union of visibility of all register objects:

$$\begin{aligned} \prec &\triangleq (R_1.\prec) \cup \cdots \cup (R_m.\prec) \\ \lhd &\triangleq (\sqsubset \cup \prec)^+ \end{aligned}$$

Since each register satisfies Property (V.1) and each individual $\prec$ only relates events from the same register, it follows that the combined $\prec$ also satisfies Property (V.1). By Lemma 2.1, the top-level $\lhd$ is then a partial order. The combination similarly satisfies Properties (M.1) to (M.4). We will use these top-level definition when relating rep events of distinct objects.

2.2.5 Linearizability. We now define linearizability formally in terms of visibility relations.

Definition 2.2. History E is linearizable with respect to data structure $\mathcal{D}$ if there exists a visibility relation $\prec$ and a total order $<$ on the set of events $E_c = \{e \mid \exists e' \in \mathcal{T}(E).\ e \prec^* e'\}$, such that: (1) $\lhd = (\sqsubset \cup \prec)^+ \subseteq <$ on E_c, and (2) executing the events in E_c in the order of $<$ is a legal sequential behavior of $\mathcal{D}$. The order $<$ is the *linearization order* (or *linearization*, for short) of E. Algorithm (or structure) $\mathcal{A}$ is linearizable wrt. $\mathcal{D}$ if every history of $\mathcal{A}$ is linearizable wrt. $\mathcal{D}$.

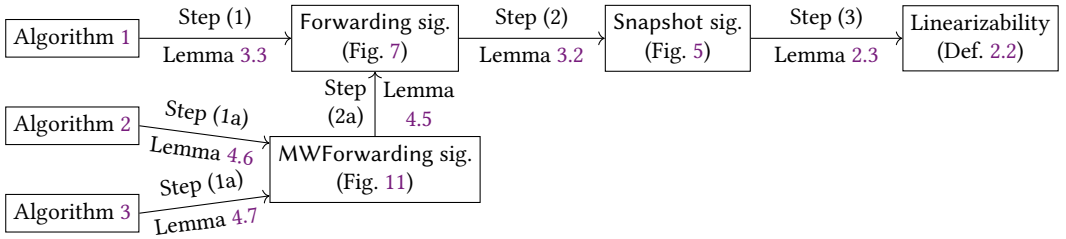


Fig. 4. Overview of the structure of the linearizability proof for each of Jayanti's snapshot algorithms.

Definition 2.2 differs somewhat from the original one [Herlihy and Wing 1990] in that we use the visibility relation $\prec$ to complete $\mathcal{T}(E)$ into E_c with events that have been observed, but have not yet terminated. The original definition existentially abstracts over the set of completing events; formally, it does not organize them into a visibility relation, but in practice these events are always added because they have been observed by some terminated event, and are necessary to ensure legal sequential behavior. The requirement $\lhd \subseteq <$ means that $<$ must respect $\sqsubset$ (in addition to $\prec$), and in particular that $<$ can reorder only overlapping events. In this paper, $\mathcal{D}$ is the snapshot data structure that has the following sequential behavior over its state (the array A):

- $w_i \in \textsc{write}_i$ writes w_i.in to $\mathsf{A}[i]$ and returns nothing, i.e., w_i.out $= ()$.
- $s \in \textsc{scan}$ does not modify A and returns a copy of A, i.e., s.out $= \mathsf{A}$.

2.3 Hierarchical Structure of the Proof

Ultimately, our goal is to prove that each of Jayanti's three snapshot algorithms is linearizable. To accomplish this in a way that maximizes proof reuse, we divide our proofs into multiple segments, which we illustrate in Fig. 4. The proof of Jayanti's first algorithm is split into the following steps:

(1) Jayanti's single-writer/single-scanner algorithm (Algorithm 1) consists of the rep events described in Fig. 6 and satisfies the forwarding snapshot signature that we give in Fig. 7.
(2) Any algorithm consisting of the rep events described in Fig. 6 and satisfying the forwarding snapshot signature from Fig. 7, also satisfies the general snapshot signature in Fig. 5.
(3) Any algorithms satisfying the general snapshot signature in Fig. 5 is linearizable.

For Jayanti's remaining algorithms, we incorporate the following two additional steps.

(1a) Jayanti's two multi-writer algorithms (Algorithms 2 and 3) consist of the rep events described in Fig. 10 and satisfy the multi-writer forwarding snapshot signature in Fig. 11.
(2a) Any algorithm consisting of the rep events described in Fig. 10 and satisfying the multi-writer forwarding snapshot signature in Fig. 11, also satisfies the forwarding snapshot signature in Fig. 7. The linearizability then follows by steps (2) and (3) above, which are thus reused and shared by all three algorithms.

We proceed to describe the intuition behind the steps (1) to (3), together with the forwarding and snapshot signatures. The detailed proofs of (1) to (3) are in Section 3. The description and proofs of (1a) and (2a) are in Section 4.

2.4 Snapshot Signature

Going backwards, we start with the general snapshot signature (Fig. 5). It describes the axioms for reasoning about the snapshot data structure as a whole, and will interface the linearizablity proofs by the following lemma, which we will prove in Section 3.

Lemma 2.3. *Histories satisfying the* Snapshot *signature (Fig. 5) are linearizable.*

signature Snapshot $\triangleq$		**Snapshot data-structure**
Let	W_i	Set of all writes of cell i of the snapshot in history
	S	Set of all scans of the snapshot in history
$\sum$	$\mathbb{W}_i \subseteq W_i$	Set of effectful writes of index i
	$\prec \subseteq (\bigcup_i \mathbb{W}_i \cup S)^2$	Visibility relation
	$\xrightarrow{\text{rf}} \subseteq \prec \cap \bigcup_i \mathbb{W}_i \times S$	Reads-from visibility
Let	$\lhd \triangleq (\sqsubset \cup \prec)^+$	Happens-before order

$$\forall e, e'.\quad e \prec^+ e' \implies e' \not\sqsubset e \tag{V.1}$$

$$\forall i, s \in \mathcal{T}(S).\quad \exists w_i.\ w_i \xrightarrow{\text{rf}} s \wedge w_i.\text{in} = s.\text{out}[i] \tag{S.1}$$

$$\forall w_i, s.\quad w_i \xrightarrow{\text{rf}} s \implies \nexists w_i'.\ w_i \lhd w_i' \lhd s \tag{S.2}$$

$$\forall w_i, w_i', s.\quad w_i \xrightarrow{\text{rf}} s \wedge w_i' \xrightarrow{\text{rf}} s \implies w_i = w_i' \tag{S.3}$$

$$\forall w_i, w_i' \in \mathbb{W}_i.\quad w_i \neq w_i' \implies w_i \lhd w_i' \vee w_i' \lhd w_i \tag{S.4}$$

$$\mathcal{T}(W_i) \subseteq \mathbb{W}_i \tag{S.5}$$

$$\forall w_i, w_i', w_j, w_j', s, s'.\quad w_i, w_j \xrightarrow{\text{rf}} s \wedge w_i', w_j' \xrightarrow{\text{rf}} s' \wedge w_i \lhd w_i' \implies w_j' \not\lhd w_j \tag{S.6}$$

Fig. 5. Signature representation of the properties of a snapshot data-structure.

The Snapshot signature is almost identical to that for atomic registers (Fig. 3); the main distinction is that we have multiple sets of writes into multiple pointers, scans (observing multiple writes) instead of reads, the sets of *effectful* writes $\mathbb{W}_i$, and the extra Properties (S.5) and (S.6).

Not all writes are immediately observable once they have started, therefore we introduce the set of effectful writes $\mathbb{W}_i$ for each i. These are the writes that are available for scans to observe (by definition of $\xrightarrow{\text{rf}}$) and which we can order by Property (S.4). Property (S.5) encodes that each terminated write must have been effectful.

Property (S.6) imposes a form of monotonicity on the ordering of the writes observed by scans over multiple memory cells. More specifically, if we have two writes w_i, w_j observed by scan s, and two writes w_i', w_j' observed by scan s', then the writes into i and j cannot be ordered in the opposite way. The property ensures that the scans can be totally ordered in an eventual linearization order. Indeed, if $w_i \lhd w_i'$ and $w_j' \lhd w_j$, and we ordered s before s', then we reach a contradiction by Property (S.2), because the event w_j occurs between w_j' and the scan s' that observes it. Similar argument applies if we try to order the scans the other way around.

2.5 Forwarding Signature

The Snapshot signature in Fig. 5 captures the key properties of snapshot algorithms, and we shall prove linearizability solely out of the axioms of this signature. However, Jayanti's algorithms have more in common than merely being snapshot algorithms, as they share the same design principle of forwarding. We can thus further modularize the proofs by axiomatizing forwarding itself.

To see what the axiomatization should accomplish, we must first discuss the high-level differences between Jayanti's algorithms that the axiomatization must abstract over. We present Algorithms 2 and 3 in detail in Section 4, but for now it suffices to know that in order to support the multi-writer functionality of Algorithms 2 and 3, we need to allow for different kinds of forwarding that go beyond simply writing into the auxiliary array B that Algorithm 1 does.

Additionally, to support multi-scanner functionality of Algorithm 3, we need to allow for an abstract notion of *virtual* scan. While in a multi-scanner setting several physical scans may run

resource A : **array**$[n]$ **of** AReg	Main memory array
event signature WRITE$_i$ $\triangleq$	**Write into cell** i
a : A$[i].W$	Rep event of writer writing value into A$[i]$
event signature VSCAN $\triangleq$	**Virtual scan**
B : **array**$[n]$ **of** AReg	Forwarding array of this virtual scan
r_i : B$[i].W$	Rep event of scanner resetting B$[i]$ by writing $\perp$
a_i : A$[i].R$	Rep event of scanner reading from A$[i]$
b_i : B$[i].R$	Rep event of scanner reading from B$[i]$

Fig. 6. Resources and event signatures for abs writes and virtual scans corresponding to Forwarding signature.

concurrently, the key idea of Jayanti for Algorithm 3 is that these scans collaborate to create a virtual scan (and a corresponding snapshot), of which at most one may exist at any given moment. Thus, even though Algorithm 3 is physically a multi-scanner, in Section 4.3 we will still be able to conceptually see it as a single-scanner one.

While rep and abs events originate from execution histories, the virtual scans are artificial events that the human verifier creates themselves for purposes of verification. In analogy with the concept of "ghost state" that is frequently used in verification of concurrent programs, we can say that virtual scans are "ghost" events. We will further require a mapping that sends each abs scan to a virtual scan to which the abs scan contributed. Thus, in a multi-scanner algorithm, a number of abs scans may be mapped to the same virtual scan. The mapping may be partial because some abs scan may not have yet reached a point for which the virtual scan representing it has been determined. For single-scanner algorithms, this map is a (total) bijection that identifies virtual and abs scans. In particular, the intuition about abs scans from Algorithm 1 will suffice to understand virtual scans in our axiomatization of forwarding principles. In the rest of the text, we use the color green to visually mark the virtual scans.

2.5.1 Event Signatures. For the Forwarding signature, it is not enough to just consider properties over the abs events, as for the Snapshot signature, but we also need to consider their rep events. For this reason, we introduce a special kind of signature, called *event signature*, to encode which rep events an abs or virtual event consists of. In particular, for an abs event e, we write $e \in \text{SIG}$, if e's rep events are enumerated by the fields of the signature SIG. If $e \in \text{SIG}$, where SIG contains the field a, we write ea for the projection $e.a$. Similarly, if e is a virtual event. We shall further assume the following properties of structures satisfying event signatures:

(ES.1) Let e be an event in a signature with field a. If e is an abs event, then ea.parent $= e$, and thus also $ea \subseteq e$, by our assumption on event structure in Section 2.2.1. If e is a virtual event, we do not insist on ea.parent $= e$. Virtual event e is a custom collection of rep events, whose parents remain the abs events that invoked them, not the collection that is e. Nevertheless, we still require $ea \subseteq e$.

(ES.2) Any event in an event signature instance is unique to the instance, i.e., if $e, e' \in \text{SIG}$, and SIG contains the field a, and $ea = e'a$, then $e = e'$.

(ES.3) Instances of event signatures may have some (or all) of their subevents undefined, unless the instance is a terminated event, in which case all its subevents must be defined. For example, if e is non-terminated, ea may be undefined. This corresponds to ongoing abs events having only a subset of its rep events executed so far.

signature Forwarding $\triangleq$			**Snapshot data-structure with forwarding**
Let	W_i	Set of all writes of cell i of the snapshot in history with each $w_i \in \textsc{write}_i$	
	S	Set of all scans of the snapshot in history	
$\sum$	Σ		Set of virtual scans with each $\sigma \in \textsc{vscan}$
	$[-]^\bullet : S \rightharpoonup \Sigma$		Partial mapping of abs scans into virtual scans
	$\xrightarrow{\text{fwd}} \subseteq \bigcup_i W_i \times \Sigma$		Forwarding visibility
Let	$w_i \xrightarrow{\text{rf}} \sigma$	$\triangleq (w_i a \xrightarrow{\text{rf}} \sigma a_i \wedge \sigma r_i \xrightarrow{\text{rf}} \sigma b_i) \vee w_i \xrightarrow{\text{fwd}} \sigma$	Reads-from visibility
	$w_i \xrightarrow{\text{wr}} w_i'$	$\triangleq w_i a \lhd w_i' a$	Writing visibility
	$\prec$	$\triangleq \xrightarrow{\text{rf}} \cup \xrightarrow{\text{wr}}$	Visibility relation
	$\lhd$	$\triangleq (\sqsubset \cup \prec)^+$	Happens-before order

$\forall s \in \text{dom}([-]^\bullet).$	$s^\bullet \subseteq s$	(F.1)
$\forall i, s \in \mathcal{T}(S).$	$\exists w_i.\ w_i \xrightarrow{\text{rf}} s^\bullet \wedge w_i.\text{in} = s.\text{out}[i]$	(F.2)
$\forall e_r \in \mathtt{A}[i].W.$	$\exists w_i.\ w_i a = e_r$	(F.3a)
$\forall \sigma, e_r \in \sigma.\mathtt{B}[i].W.$	$e_r.\text{in} = \bot \iff \exists \sigma'.\ \sigma' r_i = e_r$	(F.3b)
$\forall \sigma, \sigma' \in \Sigma.$	$\sigma \neq \sigma' \implies \sigma \sqsubset \sigma' \vee \sigma' \sqsubset \sigma$	(F.4a)
$\forall \sigma \in \Sigma.$	$\sigma r_i \sqsubset \sigma a_i \sqsubset \sigma b_i$	(F.4b)
$\forall w_i, w_i', \sigma.$	$w_i \xrightarrow{\text{fwd}} \sigma \wedge w_i' \xrightarrow{\text{fwd}} \sigma \implies w_i = w_i'$	(F.5a)
$\forall \sigma.$	$\mathcal{T}(\sigma b_i) \wedge \neg(\sigma r_i \xrightarrow{\text{rf}} \sigma b_i) \implies \exists w_i.\ w_i \xrightarrow{\text{fwd}} \sigma$	(F.5b)
$\forall w_i, \sigma.$	$w_i \xrightarrow{\text{fwd}} \sigma \implies w_i a \lhd \sigma b_i \wedge \neg(\sigma r_i \xrightarrow{\text{rf}} \sigma b_i)$	(F.5c)
$\forall w_i, \sigma.$	$\sigma r_i \xrightarrow{\text{rf}} \sigma b_i \wedge w_i \sqsubset \trianglelefteq \xrightarrow{\text{rf}} \sigma \implies w_i a \lhd \sigma a_i$	(F.6)
$\forall w_i, w_i', \sigma.$	$w_i \xrightarrow{\text{fwd}} \sigma \wedge w_i' a \lhd \sigma r_i \implies w_i \not\lhd w_i'$	(F.7a)
$\forall w_i, w_i', \sigma.$	$w_i \xrightarrow{\text{fwd}} \sigma \wedge w_i' \sqsubset \trianglelefteq \xrightarrow{\text{rf}} \sigma \implies w_i \not\lhd w_i'$	(F.7b)

Fig. 7. Signature representation of the properties of a snapshot data-structure that uses forwarding.

We now present Fig. 6 which consists of the resources and event signatures necessary for the Forwarding signature. First, we have the array A which corresponds to the main memory that a forwarding snapshot algorithm operates over. Algorithm 1 clearly has such an array. Next, we have that every abs write w_i has a unique rep event $w_i a$ signifying the physical act of writing into A. This is clearly true of Algorithm 1, as we have seen. For a virtual scan σ, the figure postulates an array B, as well as three rep events σr_i, σa_i, and σb_i, corresponding to resetting $\mathtt{B}[i]$, reading from $\mathtt{A}[i]$, and reading from $\mathtt{B}[i]$, respectively. In Algorithm 1, we have already described these events under the names sr_i, sa_i, and sb_i, and similar events will exist for Algorithms 2 and 3 as well. The notation implies that the array B and the subevents are local fields of σ. We will make use of the locality of B in the multi-scanner case, where different virtual scans have different forwarding arrays. In the case of single-scanner algorithms, the virtual and abs scans coincide, and the forwarding array of each virtual scan is instantiated with the global forwarding array.

2.5.2 Forwarding Signature Formally. Our axiomatization of forwarding principles is given in Fig. 7. The signature, which we refer to as Forwarding, declares as inputs the sets of write events W_i (for each i), and the set of scan events S. It then postulates the existence of a set of virtual scans Σ, with a mapping $[-]^\bullet$ from abs to virtual scans, and a relation $\xrightarrow{\text{fwd}}$ for visibility by forwarding that captures abstractly when a virtual scan observes a writer by forwarding. As before, these are

abstract concepts, known to satisfy only the properties listed in the scope of Σ. We will shortly provide the intuition for the axioms in Fig. 7, but first, let us enumerate how the axiomatization abstracts from Algorithm 1, so that the same signature applies to all three algorithms.

- The axiomatization does not assume that we have rep write events for forwarding ($w_i b$). Instead, it makes forwarding abstract by tying it to the relation $\xrightarrow{\text{fwd}}$, allowing for multiple different forwarding methods.
- The axiomatization links virtual scans to abs scans by the function $[-]^\bullet$, to support multiple scanners. Physical scans can overlap in time, as long as the virtual scans do not.
- The axiomatization does not assume any rep events operating with memory cell X, or for that matter, that such a cell even exists. Since X is solely used to communicate to writers if and when to forward, we instead encode the conditions for forwarding in the axioms. This is necessary because the three algorithms decide differently on whether to forward or not.

2.5.3 Intuition Behind the Forwarding Signature. Out of the forwarding visibility relation $\xrightarrow{\text{fwd}}$, we define two additional visibility relations over abstract and virtual events: $\xrightarrow{\text{rf}}$ and $\xrightarrow{\text{wr}}$ as shown in Fig. 7. These relations define what we will use as an abstract notion of a virtual scan observing a writer directly or by forwarding ($\xrightarrow{\text{rf}}$), and a writer observing a prior write ($\xrightarrow{\text{wr}}$), respectively. In each algorithm, the $\xrightarrow{\text{fwd}}$ relation will be defined differently, but the definitions of $\xrightarrow{\text{rf}}$ and $\xrightarrow{\text{wr}}$ are constant in terms of $\xrightarrow{\text{fwd}}$. We next explain each of them.

$$w_i \xrightarrow{\text{rf}} \sigma \quad \triangleq \quad (w_i a \xrightarrow{\text{rf}} \sigma a_i \wedge \sigma r_i \xrightarrow{\text{rf}} \sigma b_i) \vee w_i \xrightarrow{\text{fwd}} \sigma$$

Read visibility ($\xrightarrow{\text{rf}}$) captures that a virtual scan σ either reads the value of w_i by forwarding (disjunct $w_i \xrightarrow{\text{fwd}} \sigma$) or directly. The direct read requires that the scanner finds the value in $\mathtt{A}[i]$ (conjunct $wra_i \xrightarrow{\text{rf}} \sigma a_i$) and that the scanner finds $\perp$ for the forwarded value in $\mathtt{B}[i]$ (conjunct $\sigma r_i \xrightarrow{\text{rf}} \sigma b_i$).

$$w_i \xrightarrow{\text{wr}} w_i' \quad \triangleq \quad w_i a \lhd w_i' a$$

Write visibility ($\xrightarrow{\text{wr}}$) orders the abs writers of a common memory cell, and is defined by the ordering of underlying rep writers $w_i a$. Ordering abs writers this way works because the rep write is the unique point where the abs writer communicates its value to the data structure. This holds for all three algorithms including the multi-writer variants.

We next explain the forwarding properties. These are divided into three groups, as shown in Fig. 7. The first group consists of the structural properties related to writers and scanners.

$$\forall s \in \mathrm{dom}([-]^\bullet). \qquad s^\bullet \subseteq s \qquad \text{(F.1 revisited)}$$
$$\forall i, s \in \mathcal{T}(S). \qquad \exists w_i.\ w_i \xrightarrow{\text{rf}} s^\bullet \wedge w_i.\mathsf{in} = s.\mathsf{out}[i] \qquad \text{(F.2 revisited)}$$

Property (F.1) states that each virtual scan must be inside the interval of the abs scan it represents. Property (F.2) states that a terminated abs scan s must have a virtual scan representative that, for each index i, observed some write w_i such that the value written by w_i is the one the scan s returned for i. Property (F.2) is similar to Property (S.1), and essentially says that the observing scan correctly reads the array. On the other hand, Property (F.1) is similar to a property of linearization points, whereby a linearization point must reside within the interval of the considered event. Here instead, we have a whole virtual scan $s^\bullet$ representing when s logically occurred.

$$\forall e_r \in \mathtt{A}[i].W. \qquad \exists w_i.\ w_i a = e_r \qquad \text{(F.3a revisited)}$$

Property (F.3a) ensures that the only rep event that can write into $\mathtt{A}[i]$ is $w_i a$ of some abs write w_i. This means that only abs writes into cell i can have a rep event for writing into $\mathtt{A}[i]$; scans and writes into a cell different from i cannot, since if they did, they would have to be equal to some w_i

as per Property (ES.2) of event signatures. Also, an abs write can write into $\mathtt{A}[i]$ at most once, since any other write must equal $w_i a$.

$$\forall \sigma, e_r \in \sigma.\mathtt{B}[i].W. \qquad e_r.\mathsf{in} = \bot \iff \exists \sigma'.\ \sigma' r_i = e_r \qquad \text{(F.3b revisited)}$$

The easiest way to explain Property (F.3b) is to consider its instantiation for Algorithm 1, where virtual scanners are abs scanners, which, moreover, all share the same array B.

$$\forall e_r \in \mathtt{B}[i].W. \qquad e_r.\mathsf{in} = \bot \iff \exists \sigma'.\ \sigma' r_i = e_r \qquad \text{(F.3b')}$$

The simplified Property (F.3b') says that the only rep event that can write $\bot$ into $\mathtt{B}[i]$ is $\sigma' r_i$ of some abs scan σ'. Thus, no event except scanners can write $\bot$ into $\mathtt{B}[i]$, and the writing can be done at most once. Additionally, the rep event σr_i can *only* write $\bot$ into $\mathtt{B}[i]$. Property (F.3b) generalizes Property (F.3b') by allowing for virtual scanners, each of which has their own B array.

$$\forall \sigma, \sigma' \in \Sigma. \qquad \sigma \neq \sigma' \implies \sigma \sqsubset \sigma' \vee \sigma' \sqsubset \sigma \qquad \text{(F.4a revisited)}$$

Property (F.4a) captures that virtual scans never overlap, and are thus totally ordered. This property holds for single-scanner algorithms by assumption, but must be proved in the multi-scanner case.

$$\forall \sigma \in \Sigma. \qquad \sigma r_i \sqsubset \sigma a_i \sqsubset \sigma b_i \qquad \text{(F.4b revisited)}$$

Property (F.4b) simply reflects that the rep events of initializing the scan for element i, reading $\mathtt{A}[i]$, and then closing the scanning for element i by reading $\mathtt{B}[i]$, are invoked sequentially in the code of SCAN. This is readily visible in Algorithm 1 and remains true in Algorithms 2 and 3.

The second group are the structural properties of the forwarding visibility.

$$\forall w_i, w_i', \sigma. \qquad w_i \xrightarrow{\mathsf{fwd}} \sigma \wedge w_i' \xrightarrow{\mathsf{fwd}} \sigma \implies w_i = w_i' \qquad \text{(F.5a revisited)}$$
$$\forall \sigma. \qquad \mathcal{T}(\sigma b_i) \wedge \neg(\sigma r_i \xrightarrow{\mathsf{rf}} \sigma b_i) \implies \exists w_i.\ w_i \xrightarrow{\mathsf{fwd}} \sigma \qquad \text{(F.5b revisited)}$$
$$\forall w_i, \sigma. \qquad w_i \xrightarrow{\mathsf{fwd}} \sigma \implies w_i a \lhd \sigma b_i \wedge \neg(\sigma r_i \xrightarrow{\mathsf{rf}} \sigma b_i) \qquad \text{(F.5c revisited)}$$

Property (F.5a) captures that a scan can observe at most one forwarded write for a given index i. Property (F.5b) says that a forwarding of some write of cell i will reach the virtual scan σ if σb_i did not observe the writing of $\bot$ in $\mathtt{B}[i]$ by σr_i. Contrapositively, $\sigma r_i \xrightarrow{\mathsf{rf}} \sigma b_i$ holds if no write of cell i was forwarded to σ. Property (F.5c) states the dual implication direction of Property (F.5b) and that a forwarded write w_i must have written into $\mathtt{A}[i]$ by $w_i a$ before the scanner σ read the forwarded value by σb_i. This property captures that writers must first write their value directly before attempting to forward.

The third group are the properties that capture the forwarding principles of Jayanti. We have originally discovered these properties by extracting the common patterns from our proofs of the snapshot signature for the three algorithms, and only afterwards discovered that they actually correspond quite closely to Jayanti's forwarding principles. To describe how our axioms capture the forwarding principles, we state the principles below in English, verbatim as Jayanti does, but using our notation. We also use virtual scans instead of abstract scans to make the connection to multi-scanner algorithm direct; Jayanti only stated the principles in terms of Algorithm 1. As we shall see, our axiomatization modifies the principles slightly to encompass Algorithms 2 and 3. We also highlight and number subsentences so that we can relate them to our axioms in discussion.

(1) Suppose that [(i)]a scan operation σ misses a write operation w_i writing v because σ reads $\mathtt{A}[i]$ before w_i writes in $\mathtt{A}[i]$. If [(ii)]w_i completes before σ performs s_{off}, then [(iii)]w_i will have surely informed σ of v by writing v in $\mathtt{B}[i]$.

(2) Suppose that [(iv)]a scan operation σ reads at σb_i a non-$\bot$ value in $\mathtt{B}[i]$ by some write operation w_i. Then (a) [(v)]w_i is concurrent with σ, and (b) If [(vi)]w_i' is any write operation that is executed after w_i, then [(vii)]w_i' completes only after σ performs s_{off}.

We start with Property (F.6), which relates to Principle (1) as follows.

$$\forall w_i, \sigma. \qquad \sigma r_i \xrightarrow{\mathrm{rf}} \sigma b_i \wedge w_i \sqsubset \lessdot \xrightarrow{\mathrm{rf}} \sigma \implies w_i a \lhd \sigma a_i \qquad \text{(F.6 revisited)}$$

Here, $\sigma r_i \xrightarrow{\mathrm{rf}} \sigma b_i$ corresponds to the negation of Statement (iii), $w_i \sqsubset \lessdot \xrightarrow{\mathrm{rf}} \sigma$ corresponds to Statement (ii), and $w_i a \lhd \sigma a_i$ corresponds to the negation of Statement (i), altogether combining into an equivalent, by contraposition, of Principle (1). We now explain each correspondence individually.

- $\sigma r_i \xrightarrow{\mathrm{rf}} \sigma b_i$ states that nothing was forwarded to σ, because the event σb_i of reading $\mathtt{B}[i]$ observes the event σr_i of clearing $\mathtt{B}[i]$. This is a slightly stronger statement than the negation of Statement (iii), namely, that the write w_i was not forwarded to σ. The difference between these statements will be covered by Property (F.7b), which will handle the scenario where another write, not w_i, is forwarded to σ.
- $w_i \sqsubset \lessdot \xrightarrow{\mathrm{rf}} \sigma$, where $\sqsubset \lessdot \xrightarrow{\mathrm{rf}}$ is the relational composition of $\sqsubset$ and $\lessdot$ and $\xrightarrow{\mathrm{rf}}$, roughly translates to "w_i terminates before some other writer (possibly writing to a different memory cell) that in turn was observed by σ". We know such other writer exists, since the domain of $\xrightarrow{\mathrm{rf}}$, the last relation in the composition, consists only of writers. While this looks nothing like Statement (ii), both statements imply that w_i must have been forwarded if the writing to $\mathtt{A}[i]$ was missed. We prove this with different proofs for each algorithm, which we give in the appendices,[1] with Lemma C.2 for Algorithm 1 and Lemma D.2 for Algorithms 2 and 3. We use the statement $w_i \sqsubset \lessdot \xrightarrow{\mathrm{rf}} \sigma$, instead of potentially another statement involving s_{off} (as Jayanti's original statement does), because it makes the forwarding signature export fewer rep events, thus making it a bit more parsimonious.
- $w_i a \lhd \sigma a_i$ states that $w_i a$ wrote to $\mathtt{A}[i]$ before σa_i read from it, which directly corresponds to the negation of Statement (i).

For Principle (2a), we need to further deviate from Jayanti's original formulation, since the latter is too specific to Algorithm 1, and does not scale as-is to the multi-writer algorithms. The point of this principle is to ensure that a writer may only forward values that are, intuitively, "current". The principle itself ensures this property, but only in the single-writer case of Algorithm 1. Indeed, in Algorithm 1, a writer only forwards its own value (and other writes to the same pointer are prohibited by assumption); therefore, if a write is concurrent to the scan, then it can only forward a value that is current. But in a multi-writer case, as we shall see in Section 4, we want to allow a writer to forward values of other writers. In particular, a writer that writes and then stalls indefinitely could still be forwarded by another write to some concurrent scan. But we still need to ensure that the forwarded write is not arbitrarily old. We do so by insisting that a write w_i forwarded to σ cannot occur before a write w_i' which wrote to $\mathtt{A}[i]$ with $w_i a$ before σ started, i.e., executed σr_i. In other words, we require that the forwarded write w_i does not "happen before" any write w_i' that started before the start of the scan. A write w_i' has a *newer* value, so its existence should prevent w_i from being forwarded. We formalize this as Property (F.7a).

$$\forall w_i, w_i', \sigma. \qquad w_i \xrightarrow{\mathrm{fwd}} \sigma \wedge w_i' a \lhd \sigma r_i \implies w_i \not\lhd w_i' \qquad \text{(F.7a revisited)}$$

This captures that writers must forward to scan σ either the latest write that wrote to $\mathtt{A}[i]$ before σr_i, or some write that occurred after σr_i, thus ensuring that the forwarded write is current.

Lastly, Property (F.7b) relates to the contrapositive of Principle (2b).

$$\forall w_i, w_i', \sigma. \qquad w_i \xrightarrow{\mathrm{fwd}} \sigma \wedge w_i' \sqsubset \lessdot \xrightarrow{\mathrm{rf}} \sigma \implies w_i \not\lhd w_i' \qquad \text{(F.7b revisited)}$$

Again, $w_i \xrightarrow{\mathrm{fwd}} \sigma$ directly corresponds to Statement (iv). Formal statement $w_i' \sqsubset \lessdot \xrightarrow{\mathrm{rf}} \sigma$ corresponds to the negation of Statement (vii), for the same reason as in Property (F.6), since (ii) is the negation of (vii). Finally, $w_i \not\lhd w_i'$ directly corresponds to the negation of (vi), with the exception that we use

[1] All appendices are in the supplementary materials section of the ACM Digital Library.

$\lhd$ instead of Jayanti's "executed after" to relate the writes w_i and w_i'. The two correspond in the single-writer case, but our statement extends to the multi-writer case as well.

Algorithm 1 satisfies this set of properties of the Forwarding signature, and they are sufficient to prove that an algorithm is a snapshot algorithm according to the Snapshot signature, which is in turn sufficient to prove linearizability. In Section 3 we sketch why these claims holds, and in Section 4 we sketch how Algorithms 2 and 3 satisfy the Forwarding signature. The complete proofs are in the appendices in the supplementary materials section of the ACM Digital Library.

3 PROOF SKETCHES FOR ALGORITHM 1

We now sketch the proofs of our signatures implying linearizability, to illustrate the gist of reasoning by visibility. We begin by showing how histories that satisfy the Snapshot signature are linearizable. This proof is based on a similar proof by Chakraborty et al. [2015] for queues.

Lemma 2.3. *Histories satisfying* Snapshot *signature (Fig. 5) are linearizable.*

Proof sketch. Following Definition 2.2 of linearizability, we start by choosing a visibility relation $\prec$ that constructs the set $E_c = \{e \mid \exists e' \in \mathcal{T}(E).\ e \prec^* e'\}$, and then extend it to a linearization $<$. Let $\prec$ be the visibility relation obtained by restricting the visibility relation postulated by Snapshot to $\mathcal{T}(S) \cup \bigcup_i \mathbb{W}_i$. Events outside of the restriction are non-terminated scans or writes that have not executed their effect yet; they do not modify the abstract state, and are hence not necessary for linearization. Additionally, by Property (S.5) we know that every terminated write is effectful. Thus going forward, we consider $E_c = \mathcal{T}(S) \cup \bigcup_i \mathbb{W}_i$.

We next extend the happens-before order $\lhd$ on the selected set of events as follows. The idea is to keep extending this order with more relations between the events of the selected set until we reach a total order, which will be the desired linearization order. We first induce a helper order

$$\lhd_w \quad \triangleq \quad (\lhd \ \cup \ \xrightarrow{\text{wrDiff}})^+$$

which ranges over writes (into different cells), where $w_i \xrightarrow{\text{wrDiff}} w_j'$ holds if $i \neq j$ and there exists a write w_j and a scan s such that $w_i, w_j \xrightarrow{\text{rf}} s$ and $w_j \lhd w_j'$. Or, in English, we order w_i before w_j', if w_i is observed by some scan along with w_j, and w_j is ordered before w_j'. The $\lhd_w$ order is a partial order by Properties (S.1) to (S.4) and (S.6) (see Lemma A.2 from Appendix A for the complete proof).

Next, we select an arbitrary total order $<_w$ over writes that extends $\lhd_w$. Since the set of selected events is finite, such a total order always exists by Zorn's Lemma. We use the order $<_w$ to determine which event we will consider as being last in the eventually constructed linearization order. We say that a write is latest-in-time if it is the greatest in $<_w$, while a scan is latest-in-time if it is maximal in $\lhd$ and all the writes that it observes are the greatest in $<_w$ for their respective memory cell. As long as the set of events is non-empty, there will exist a latest-in-time event under this definition by Properties (S.1) to (S.4) (see Lemma A.4 from Appendix A for the complete proof). By inductively selecting the latest-in-time event, we construct a total order of events that is consistent with $\lhd$ and snapshot semantics, and is thus a linearization order. More concretely, we add the latest-in-time event to the end of the linearization order, forming a chain with previously added events, and repeat this step with the set of events excluding the last selected. □

Next, we show that histories satisfying the Forwarding signature also satisfy the Snapshot signature. Because some Forwarding properties use rep events, to be able to efficiently use them, we first need a lemma that turns a $\lhd$ relation between virtual or abs events into a $\lhd$ relation between rep events. The lemma will also be useful in showing that Algorithm 1 (and Algorithms 2 and 3 in Section 4) satisfy the Forwarding signature, once Properties (F.4b) and (F.5c) have been proven.

Lemma 3.1 (Happens-before of subevents). *Assume Properties* (F.4b) *and* (F.5c). *For events* $e, e' \in \bigcup_i W_i \cup \Sigma$ *of the* Forwarding *signature (Fig. 7) where* e' *is populated with at least one rep event from its event structure (Fig. 6), if* $e \lhd e'$ *holds (where* $\lhd$ *follows the definition from Fig. 7) then there exists* e_r *and* e'_r *where* $e_r \lhd e'_r$ *holds, with* e_r *and* e'_r *belonging to* e *and* e' *respectively.*

Proof. We can view $e \lhd e'$ as a chain of $\lhd_1$ relations connecting e and e'; that is, $e \lhd_1 \cdots \lhd_1 e'$, where by definition $\lhd_1 = (\sqsubset \cup \prec) = (\sqsubset \cup \xrightarrow{\text{wr}} \cup \xrightarrow{\text{rf}})$. It is easy to see that each abs event in this chain is populated with a rep event. Indeed, if an abs event's relation $\lhd_1$ to its successor is realized by $\sqsubset$, then the event is terminated and all of its rep events are executed (and by our axioms, each abs event must have at least one rep event). Alternatively, if the abs event's relation $\lhd_1$ to its successor is realized by $\prec$, then the event must be populated as per the definitions of reads-from and writing visibility in Fig. 7, and in the $\xrightarrow{\text{fwd}}$ case by Property (F.5c). This leaves out e', which has no successor, but e' is populated by assumption.

Now the proof is by induction on the length of the above chain. The base case is when $e \lhd_1 e'$. If $e \sqsubset e'$, the proof is by Property (RB.2) giving us $e_r \sqsubset e'_r$ for arbitrary e_r and e'_r belonging to e and e' respectively. If $e \xrightarrow{\text{wr}} e'$, the proof is by the definition of $\xrightarrow{\text{wr}}$ in Fig. 7, giving us $w_i a \lhd w'_i a$ with $e = w_i$ and $e' = w'_i$. For the last case, $e \xrightarrow{\text{rf}} e'$, we have $e = w_i$ and $\sigma' = e'$ with either $w_i a \xrightarrow{\text{rf}} \sigma' a_i$ or $w_i \xrightarrow{\text{fwd}} \sigma'$. The former case directly exhibits rep events that satisfy our goal. The latter does so as well, since by Property (F.5c), it must be $w_i a \lhd \sigma' b_i$. The inductive step is similar, with the addition that we need transitivity of $\lhd$ and Property (F.4b) to join events. □

Lemma 3.2. *Histories satisfying* Forwarding *signature (Fig. 7) also satisfy* Snapshot *signature.*

Proof sketch. We start by determining the instantiations of $\mathbb{W}_i$, $\xrightarrow{\text{rf}}$, and $\prec$ for the Snapshot signature. Since the effect of w_i is observable only after the execution of $w_i a$, we let $w_i \in \mathbb{W}_i$ iff $w_i a$ is defined. For the other notions, we first define a helper visibility over scans $\xrightarrow{\text{sc}}$, which we use to define $\prec$. The highlighted relations $\xrightarrow{\text{rf}}$ and $\xrightarrow{\text{wr}}$ originate from the Forwarding signature.

$$w_i \xrightarrow{\text{rf}} s \triangleq w_i \xrightarrow{\text{rf}} s^\bullet \qquad s \xrightarrow{\text{sc}} s' \triangleq s^\bullet \sqsubset s'^\bullet \qquad \prec \triangleq \xrightarrow{\text{rf}} \cup \xrightarrow{\text{wr}} \cup \xrightarrow{\text{sc}}$$

Property (V.1) holds since if $e \prec^+ e'$ and $e' \sqsubseteq e$, then we can construct a $\lhd$-cycle of rep events with Lemma 3.1, contradicting irreflexivity of $\lhd$ (Lemma 2.1). Property (S.1) holds directly by Property (F.2). Property (S.3) holds because by the definition of $w_i \xrightarrow{\text{rf}} \sigma$ in Forwarding signature, it is either $w_i a \xrightarrow{\text{rf}} \sigma a_i$ for which Property (M.3) ensures uniqueness, or $w_i \xrightarrow{\text{fwd}} \sigma$ where Property (F.5a) ensures uniqueness. Property (S.4) of total ordering on effectful writes follows because in Forwarding signature, $w_i \xrightarrow{\text{wr}} w'_i$ is defined as $w_i a \lhd w'_i a$, and the rep writes are totally ordered by Property (M.4). Property (S.5) that all terminated writes are effectful holds because for every terminated write w_i we must have $w_i a$ defined.

We next prove Property (S.2). We sketch the proof by assuming there is a write w'_i such that $w_i \lhd w'_i \lhd s$ where $w_i \xrightarrow{\text{rf}} s$, and then deriving a contradiction. Unfolding the definition of $\xrightarrow{\text{rf}}$ in $w_i \xrightarrow{\text{rf}} s$, we get $w_i \xrightarrow{\text{rf}} \sigma$ where $\sigma = s^\bullet$. Also, from $w'_i \lhd s$, using Property (F.1), it turns out that $w'_i \lhd \sigma$, and moreover, the latter splits into two possible cases: $w'_i \sqsubset \trianglelefteq \xrightarrow{\text{rf}} \sigma$ and $w'_i a \lhd \sigma r_i$. The full proof of how the cases follow from the premise and that they exhaust the possibilities is given in Appendix B. Note that the proposition $w'_i \sqsubset \trianglelefteq \xrightarrow{\text{rf}} \sigma$ is the key part of Properties (F.6) and (F.7b), while $w'_i a \lhd \sigma r_i$ is the key part of Property (F.7a); in fact, this exhaustion of cases of $w'_i \lhd \sigma$ is what let us rediscover and mathematically formulate the forwarding principles. The proof now proceeds by analyzing $w_i \xrightarrow{\text{rf}} \sigma$. From the definition of reads-from visibility in Fig. 7, we also get two cases: $(w_i a \xrightarrow{\text{rf}} \sigma a_i \wedge \sigma r_i \xrightarrow{\text{rf}} \sigma b_i)$ and $w_i \xrightarrow{\text{fwd}} \sigma$, for a total of four cases when combined with the above. In each case, contradiction follows easily, relying in different cases on a different subset of Properties (F.6), (F.7a), (F.7b) and (M.2).

Finally, we prove Property (S.6); that is, $w_i, w_j \xrightarrow{\text{rf}} s$ and $w_i', w_j' \xrightarrow{\text{rf}} s'$ with $w_i \lhd w_i'$ and $w_j' \lhd w_j$ lead to contradiction. As before, we first transform the assumptions $w_i, w_j \xrightarrow{\text{rf}} s$ and $w_i', w_j' \xrightarrow{\text{rf}} s'$ into $w_i, w_j \xrightarrow{\text{rf}} \sigma$ and $w_i', w_j' \xrightarrow{\text{rf}} \sigma'$, where $\sigma = s^\bullet$ and $\sigma' = s'^\bullet$. Next, by property Property (F.4a), the virtual scans are totally ordered by $\sqsubset$, thus either $\sigma \sqsubset \sigma'$ or $\sigma' \sqsubset \sigma$ (the case when $\sigma = \sigma'$ contradicts reads-from uniqueness Property (S.3)). In the first case (the second is symmetric), we have $w_j' \lhd w_j \xrightarrow{\text{rf}} \sigma \sqsubset \sigma'$, and thus also $w_j' \lhd w_j \lhd \sigma'$. In other words, we have a scan σ' observing a write w_j' with another intervening write w_j showing up in between. But this contradicts the argument we carried out for Property (S.2) above. □

Lemma 3.3. *Every execution of Algorithm 1 satisfies the* Forwarding *signature (Fig. 7).*

Proof sketch. We instantiate the set of virtual scans to be the same as the set of abs scans (since Algorithm 1 is single-scanner), and define $[-]^\bullet$ by $s^\bullet = s$. Property (F.1), requiring $s^\bullet \in s$, follows immediately because every event e satisfies $e \subseteq e$. Since Algorithm 1 is a single-scanner algorithm, the total order of virtual scans Property (F.4a) naturally follows.

We instantiate forwarding visibility $w_i \xrightarrow{\text{fwd}} s \triangleq w_i b \xrightarrow{\text{rf}} sb_i$, to directly capture how forwarding is defined for Algorithm 1. Properties (F.2), (F.3a), (F.3b) and (F.4b) hold by the structure of the algorithm, while Properties (F.5a) to (F.5c) follows from the $\xrightarrow{\text{fwd}}$ instantiation (proved in Appendix C). For the remaining Properties (F.6) to (F.7b), we need two helper lemmas:

- Lemma C.1: $s_{on} \xrightarrow{\text{rf}} w_i x$ iff we have $s_{on} \lhd w_i x$ and $s_{off} \not\lhd w_i x$.
- Lemma C.2: If $w_i' \sqsubset \trianglelefteq \xrightarrow{\text{rf}} s$ then $s_{off} \not\lhd w_i' x$ and if $w_i' b$ was executed, then $w_i' b \lhd sb_i$.

(The above lemmas are also proved in Appendix C.) We focus here on proving Property (F.6) which involves showing if $w_i' \sqsubset \trianglelefteq \xrightarrow{\text{rf}} s$ and $sr_i \xrightarrow{\text{rf}} sb_i$ then $w_i' a \lhd sa_i$. By Lemma C.2, we have $s_{off} \not\lhd w_i' b$ and $w_i' b \lhd sb_i$ if $w_i' b$ was executed. Consider if we have $s_{on} \lhd w_i' x$, then by Lemma C.1 we have $s_{on} \xrightarrow{\text{rf}} w_i' x$ which in turn implies that $w_i' b$ must have executed, however that contradicts Property (M.2) by $w_i' b$ occurring in between $sr_i \xrightarrow{\text{rf}} sb_i$, thus we must have $s_{on} \not\lhd w_i' x$. By Property (RB.1) over $w_i' a \sqsubset w_i' x$ and $s_{on} \sqsubset sa_i$, since $s_{on} \not\lhd w_i' x$ contradicts $s_{on} \sqsubset w_i' x$, we must have $w_i' a \sqsubset sa_i$, which is our goal. □

4 MULTI-WRITER ALGORITHMS

To allow for correct non-blocking algorithms with multiple concurrent writers, as well as for multiple concurrent scanners in the case of Algorithm 3, Jayanti uses **LL** (load-link), **SC** (store-conditional) and **VL** (validate) operations [Jensen et al. 1987] in addition to simple memory mutation and dereference. We thus need to extend the register signature to account for the new operations. Additionally, Algorithms 2 and 3, both being multi-writer algorithms share more structure than what we capture in the Forwarding signature of Fig. 7. We thus introduce a new MWForwarding signature to capture the commonality of the multi-writer algorithms. We then show that histories satisfying MWForwarding also satisfy Forwarding and that Algorithms 2 and 3 satisfy MWForwarding.

4.1 LL/SC Registers

For a memory cell X, **LL**(X), or load-link, reads from, and returns the value of X. It also records the time of the read, for use in future **SC** and **VL** operations of the same thread. **SC**(X, v), or store-conditional, writes v into X and returns `true` if no other write into X occurred since the most recent **LL**(X) from the same thread. Otherwise, **SC** returns `false`, keeping X unchanged. Finally, **VL**(X), or validate, returns truth values identically to **SC**, but does not mutate X.

To capture the properties of memory with the **LL**, **SC** and **VL** operations, we introduce a new signature in Fig. 8, which we call **LL**/**SC** registers, or LLReg for short. It extends AReg from Fig. 3. We will treat each execution of **LL**, **SC** and **VL** as a read-like event, i.e., they will observe some write by $\xrightarrow{\text{rf}}$. Additionally, we denote the LL visibility $\xrightarrow{\text{ll}}$ to link **LL** events with their corresponding

signature LLReg $\triangleq$		**LL/SC Register**
Let	W, R, LL, SC, VL	Set of all writes, reads, LL, SC, and VL operations
	$\mathcal{S} \subseteq SC \cup VL$	Set of all successful SC and VL operations
	$W_c = W \cup (SC \cap \mathcal{S})$	Set of all write-like events of the register
	$R_c = R \cup LL \cup SC \cup VL$	Set of all read-like events of the register
$\sum$	$\prec \subseteq (W \cup R_c)^2$	Visibility relation
	$\xrightarrow{\text{rf}} \subseteq \prec \cap W_c \times R_c$	Reads-from visibility
	$\xrightarrow{\text{ll}} \subseteq \prec \cap LL \times (SC \cup VL)$	LL visibility
Let	$\lhd \triangleq (\sqsubset \cup \prec)^+$	Happens-before order

$\forall e, e'.$	$e \prec^+ e' \implies e' \not\sqsubseteq e$	(V.1)
$\forall r \in \mathcal{T}(R_c).$	$\exists w \in W_c.\ w \xrightarrow{\text{rf}} r$	(M$^+$.1a)
$\forall w \in W_c, r \in \mathcal{T}(R \cup LL).$	$w \xrightarrow{\text{rf}} r \implies w.\text{in} = r.\text{out}$	(M$^+$.1b)
$\forall c \in \mathcal{T}(SC \cup VL).$	$c.\text{out} = \text{booleanOf}(c \in \mathcal{S})$	(M$^+$.1c)
$\forall w \in W_c, r \in R_c.$	$w \xrightarrow{\text{rf}} r \implies \nexists w' \in W_c.\ w \lhd w' \lhd r$	(M$^+$.2)
$\forall w, w', r.$	$w \xrightarrow{\text{rf}} r \wedge w' \xrightarrow{\text{rf}} r \implies w = w'$	(M$^+$.3)
$\forall w, w' \in W_c.$	$w \neq w' \implies w \lhd w' \vee w' \lhd w$	(M$^+$.4)
$\forall c \in SC \cup VL.$	$\exists l \in \mathcal{T}(LL).\ l \xrightarrow{\text{ll}} c$	(M$^+$.5)
$\forall l, c.$	$l \xrightarrow{\text{ll}} c \implies l.\text{parent} = c.\text{parent} \wedge \nexists l' \in LL.\ l \lhd l' \lhd c$	(M$^+$.6)
$\forall w, w', l, c.$	$l \xrightarrow{\text{ll}} c \wedge w \xrightarrow{\text{rf}} l \wedge w' \xrightarrow{\text{rf}} c \implies (w = w' \iff c \in \mathcal{S})$	(M$^+$.7)

Fig. 8. Signature for atomic register with **LL**/**SC**/**VL** operations.

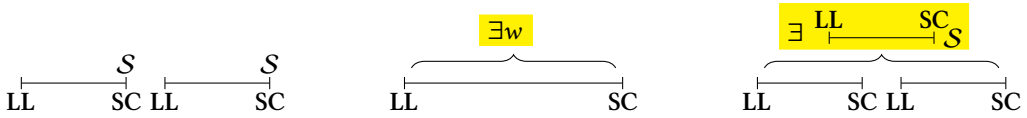
SC/**VL** events. To capture the concept of **SC**/**VL** operations being successful, we introduce the set of successful events $\mathcal{S}$. If an **SC** event is in $\mathcal{S}$, it was successful and must have written to memory, therefore we treat it as a write-like event. These sets are formally defined in the top part of Fig. 8.

Further, referring to Fig. 8, the Properties (M$^+$.1a) to (M$^+$.4) are almost identical to (M.1) to (M.4) from AReg from Fig. 3. The main difference is that we substitute quantification over writes and reads with quantification over write-like events ($W_c = W \cup (SC \cap \mathcal{S})$) and read-like events ($R_c = R \cup LL \cup SC \cup VL$), respectively. We also split Property (M.1) into three: (M$^+$.1a) says that each terminated read-like event has some observed write-like event; (M$^+$.1b) says that the output of reads and **LL** is the same as the input of the observed write-like event, and (M$^+$.1c) says that the output of **SC** and **VL** depends on its success. Additionally, Property (M$^+$.5) states that each **SC** and **VL** event has a terminated **LL** event it is related to in $\xrightarrow{\text{ll}}$. Property (M$^+$.6) says that each $\xrightarrow{\text{ll}}$ pair has the same abs parent, capturing that the two events of the pair were executed in the same thread (as per our assumption in Section 2.2.1 that each event is single-threaded), and that no other **LL** event occurs in between the pair. Finally, Property (M$^+$.7) captures that successful events observe the same write-like event as their $\xrightarrow{\text{ll}}$-related **LL** event, meaning no other modifications occurred between the **LL** event and the successful event.

There are several important properties of **LL**/**SC**/**VL** that the algorithms relies upon for synchronization. We codify them in the following three lemmas and illustrate in Fig. 9. For each of the lemmas, the events l, c, w, and variants, operate over the same memory cell X : LLReg.

Lemma 4.1. *Let $l \xrightarrow{\text{ll}} c$ and $l' \xrightarrow{\text{ll}} c'$ with $c, c' \in \mathcal{T}(SC \cap \mathcal{S})$. If $c \lhd c'$ then $c \lhd l'$.*

Proof. By Property (M$^+$.5), we know that l and l' are terminated, and by Property (M$^+$.1a), we have that each of l, l', c and c' observes some write-like event. By Property (M$^+$.7) with $c, c' \in \mathcal{S}$,



(a) Lemma 4.1: Successful **LL**/**SC** pairs do not overlap.

(b) Lemma 4.2: There will always exist some write-like event in between an **LL**/**SC** pair.

(c) Lemma 4.3: If **SC** is the only kind of write, then there exists a writing **LL**/**SC** pair in the interval of two consecutive **LL**/**SC** pairs.

Fig. 9. Illustration of key properties of **LL**/**SC**.

then l and c observe the same write-like event, and dually for l' and c', i.e., there exists w and w' such that $w \xrightarrow{\text{rf}} l$ and $w \xrightarrow{\text{rf}} c$ and $w' \xrightarrow{\text{rf}} l'$ and $w' \xrightarrow{\text{rf}} c'$. By Property (M.4), we either have $w' \lhd c$ or $c \trianglelefteq w'$. In the former case, we have $w' \lhd c \lhd c'$, which contradicts Property (M.2) by there being a c in between $w' \xrightarrow{\text{rf}} c'$. Thus, we must have the latter case, giving us $c \trianglelefteq w'$, letting us construct $c \trianglelefteq w' \xrightarrow{\text{rf}} l'$ which implies our goal $c \lhd l'$. □

Lemma 4.2. *Let $l \xrightarrow{\text{ll}} c$ with either $c \in \mathcal{T}(SC)$ or $c \in \mathcal{T}(VL \setminus \mathcal{S})$. Then there exists some $w \in W_c$ such that $w \not\trianglelefteq l$ and $w \trianglelefteq c$.*

Proof. If $c \in \mathcal{T}(SC \cap \mathcal{S})$, then c must have successfully written; thus $c = w$ trivially satisfies $w \trianglelefteq c$ and $w \not\trianglelefteq l$. The latter holds, because otherwise we get $c \lhd l$ and $l \trianglelefteq c$ and thus $c \lhd c$ which contradicts irreflexivity of $\lhd$. Next, consider $c \notin \mathcal{S}$, where $c \in \mathcal{T}(SC \cup VL)$.

By Property (M^+.5), l is terminated, and by Property (M^+.1a), for l and c there exist write-like events $w, w' \in W_c$ observed by l and c, respectively. By Property (M^+.7), these write-like events must be distinct, because otherwise c would have been successful. In other words, $w \xrightarrow{\text{rf}} l$ and $w' \xrightarrow{\text{rf}} c$ where $w \neq w'$. Taking w' to be the required write of the lemma, we trivially have $w' \trianglelefteq c$. To show that also $w' \not\trianglelefteq l$, we assume $w' \lhd l$ and derive a contradiction. By Property (M.4), we either have $w \lhd w'$ or $w' \lhd w$. In the first case, we contradict Property (M^+.2) for $w \xrightarrow{\text{rf}} l$ by $w \lhd w' \lhd l$. In the second case, we contradict Property (M^+.2) for $w' \xrightarrow{\text{rf}} c$ by $w' \lhd w \xrightarrow{\text{rf}} l \xrightarrow{\text{ll}} c$. □

Lemma 4.3. *Let $l \xrightarrow{\text{ll}} c \sqsubset l' \xrightarrow{\text{ll}} c'$ where $c, c' \in \mathcal{T}(SC)$. If within the time frame of the events l, l', c, and c' there are no mutations to register* X *except by* **SC**, *then there exist some l'' and c'' (over* X*) with $l'' \xrightarrow{\text{ll}} c''$ such that $l'' \not\trianglelefteq l$ and $c'' \trianglelefteq c'$ and $c'' \in SC \cap \mathcal{S}$.*

Proof. From the assumption, by Lemma 4.2, there exists some c_0 and c_0' such that $c_0 \not\trianglelefteq l$ and $c_0 \trianglelefteq c$ and $c_0' \not\trianglelefteq l'$ and $c_0' \trianglelefteq c'$. Since there are no other writes, we can only have $c_0, c_0' \in SC \cap \mathcal{S}$, by Property ($M^+$.5) there has to be some l_0' such that $l_0' \xrightarrow{\text{ll}} c_0'$. Let l_0' and c_0' be the existentials, we trivially have $c_0' \trianglelefteq c'$ and $c_0' \in SC \cap \mathcal{S}$. For $l_0' \not\trianglelefteq l$, assume we have $l_0' \lhd l$, we derive a contradiction. We must have $c_0 \lhd c_0'$, since we otherwise we contradict $c_0' \not\trianglelefteq l'$ by $c_0' \lhd c_0 \trianglelefteq c \sqsubset l'$, meaning we also must have $c_0 \lhd l_0'$ for c_0' to be successful, however this contradicts $c_0 \not\trianglelefteq l$ by $c_0 \lhd l_0' \lhd l$. □

Lemma 4.1 says that if we have two successful **LL**/**SC** pairs, then the intervals of the two pairs do not overlap. To see the intuition behind the other two lemmas, consider the special case when the order $\lhd$ is total (e.g., it is a linearization order), so that $\not\trianglelefteq$ is equivalent to $\trianglerighteq$. In that case, Lemma 4.2 says that an **LL**/**SC** interval, or an **LL**/**VL** interval with a failing **VL**, must contain some succesful write-like event. Similarly, Lemma 4.3 says there exists a successful **LL**/**SC** interval contained within the interval defined by two sequential **LL**/**SC** intervals, even if the latter two intervals themselves correspond to failing **SC**'s. The last property will be used by multi-writer algorithms as follows. Referring to Algorithm 2, a write procedure will try to execute forwarding by invoking two sequential intervals of **LL**/**SC** pairs into the cell $\mathtt{B}[i]$. Even if both **SC**'s fail, Lemma 4.3 guarantees

resource A : **array**$[n]$ **of** val
resource B : **array**$[n]$ **of** val $\cup \{\bot\}$
resource X : $\mathbb{B} :=$ false

1: WRITE$(i : \mathbb{N}, v :$ val$)$ $\triangleq$ ▷ w_i
2: A$[i] := v$ ▷ $w_i a$
3: $x \leftarrow$ **LL**(X) ▷ $w_i x$
4: **if** x **then**
5: FORWARD(i) ▷ $w_i f_1$
6: FORWARD(i) ▷ $w_i f_2$

7: FORWARD$(i : \mathbb{N})$ $\triangleq$ ▷ f
8: **LL**(B$[i]$) ▷ fb
9: $v \leftarrow$ A$[i]$ ▷ fa
10: $x' \leftarrow$ **VL**(X) ▷ fx
11: **if** x' **then** **SC**(B$[i], v$) ▷ fb

12: SCAN: **array**$[n]$ **of** val $\triangleq$ ▷ s
13: **for** $i \in \{0 \ldots n-1\}$ **do**
14: B$[i] := \bot$ ▷ sr_i
15: X := true ▷ s_{on}
16: **for** $i \in \{0 \ldots n-1\}$ **do**
17: $a \leftarrow$ A$[i]$ ▷ sa_i
18: $V[i] := a$
19: X := false ▷ s_{off}
20: **for** $i \in \{0 \ldots n-1\}$ **do**
21: $b \leftarrow$ B$[i]$ ▷ sb_i
22: **if** $b \neq \bot$ **then** $V[i] := b$
23: **return** V

Algorithm 2. Jayanti's multi-writer, single-scanner snapshot algorithm.

resource A : **array**$[n]$ **of** AReg
resource X : LLReg

event signature VSCAN+ $\triangleq$

B	: **array**$[n]$ **of** LLReg
r_i	: B$[i].W$
a_i	: A$[i].R$
b_i	: B$[i].R$
on	: X.W_c
$\underline{on}$	: X.W_c or X.R_c
off	: X.W_c
$\underline{off}$	: X.W_c or X.R_c

event signature WRITE$_i$+ $\triangleq$

a	: A$[i].W$
x	: X.LL
f_1, f_2	$\in$ FORWARD$_i$

event signature FORWARD$_i$ $\triangleq$

$\mathsf{B}_{\mathsf{cell}}$	: LLReg
b	: $\mathsf{B}_{\mathsf{cell}}.LL$
$\underline{a}$	: A$[i].R$
$\underline{x}$	: X.VL
$\underline{b}$	: $\mathsf{B}_{\mathsf{cell}}.SC$

Fig. 10. Resources and event signatures for abs writes, virtual scans and virtual forwarding corresponding to MWForwarding. The rep events correspond to the rep events with the same suffix in Algorithm 2, with the exception of $\underline{on}$ and $\underline{off}$ which are new and correspond to a potential observer of on and off respectively.

that another concurrent write will have invoked a successful **LL**/**SC** interval, thus forwarding the same value on behalf of the original write.

4.2 Multi-Writer Forwarding Signature

4.2.1 Description of Algorithm 2. We first discuss Algorithm 2, which will motivate the definition of MWForwarding signature. The common aspect with Algorithm 1 is that the writer communicates its value to the data structure by writing into A$[i]$. Whereas Algorithm 1 may try to communicate the same value *again* by forwarding to B$[i]$, in Algorithm 2, the forwarding procedure reads whichever value is currently present in A$[i]$ and attempts to forward it to B$[i]$ by means of **LL** and **SC**. Because multiple writes may be racing on A$[i]$, the forwarding procedure may read and forward a different value from A$[i]$ than the one the writer initially wrote. The forwarding procedure may even fail to forward anything. Nevertheless, Lemma 4.3 provides a guarantee that there will exist *some*

signature MWForwarding $\triangleq$ **Multi-Writer snapshot with forwarding**

Let	W_i	Set of all writes of cell i of the snapshot in history with each $w_i \in \text{WRITE}_i+$
	S	Set of all scans of the snapshot in history
	F_i	Set of all forwarding events in history with each $f \in \text{FORWARD}_i$
$\sum$	Σ	Set of virtual scans with each $\sigma \in \text{VSCAN}+$
	$[-]^\bullet : S \rightharpoonup \Sigma$	Partial mapping of abs scans into virtual scans
	$[-]_w : F_i \to W_i$	Write which executed the forwarding event
	$[-]_\sigma : F_i \to \Sigma$	Virtual scan observed by forwarding event

Let	$w_i \xrightarrow{\text{fwd}} \sigma \triangleq \exists f \in F_i.\ w_i a \xrightarrow{\text{rf}} f\underline{a} \wedge f\underline{b} \xrightarrow{\text{rf}} \sigma b_i$		Forwarding visibility
	$w_i \xrightarrow{\text{rf}} \sigma \triangleq (w_i a \xrightarrow{\text{rf}} \sigma a_i \wedge \sigma r_i \xrightarrow{\text{rf}} \sigma b_i) \vee w_i \xrightarrow{\text{fwd}} \sigma$		Reading visibility
	$w_i \xrightarrow{\text{wr}} w_i' \triangleq w_i a \lhd w_i' a$		Writing visibility
	$\prec \triangleq \xrightarrow{\text{rf}} \cup \xrightarrow{\text{wr}}$		Visibility relation
	$\lhd \triangleq (\sqsubset \cup \prec)^+$		Happens-before order
$\forall s \in \text{dom}([-]^\bullet).$	$s^\bullet \subseteq s$		(F.1)
$\forall i, s \in \mathcal{T}(S).$	$\exists w_i.\ w_i \xrightarrow{\text{rf}} s^\bullet \wedge w_i.\text{in} = s.\text{out}[i]$		(F.2)
$\forall e_r \in \mathsf{A}[i].W.$	$\exists w_i.\ w_i a = e_r$		(F.3a)
$\forall \sigma, e_r \in \sigma.\mathsf{B}[i].W_c.$	$e_r.\text{in} = \bot$	$\iff \exists \sigma'.\ \sigma' r_i = e_r$	(F.3b)
$\forall \sigma, e_r \in \sigma.\mathsf{B}[i].W_c.$	$e_r.\text{in} \neq \bot$	$\iff \exists f.\ f\underline{b} = e_r$	(F$^+$.3c)
$\forall \sigma, e_r \in \mathsf{X}.R_c.$	$\sigma_{on} \xrightarrow{\text{rf}} e_r$	$\iff \sigma_{on} \lhd e_r \wedge \sigma_{off} \not\lhd e_r$	(F$^+$.3d)
$\forall \sigma, \sigma' \in \Sigma.$	$\sigma \neq \sigma'$	$\implies \sigma \sqsubset \sigma' \vee \sigma' \sqsubset \sigma$	(F.4a)
$\forall \sigma \in \Sigma.$	$\sigma r_i \sqsubset \sigma_{on} \xRightarrow{\text{rf}} \sigma_{\underline{on}} \sqsubset \sigma a_i \sqsubset \sigma_{off} \xRightarrow{\text{rf}} \sigma_{\underline{off}} \sqsubset \sigma b_i$		(F$^+$.4b)
$\forall w_i \in W_i.$	$w_i a \sqsubset w_i x \sqsubset w_i f_1 \sqsubset w_i f_2$		(F$^+$.4c)
$\forall f \in F_i.$	$f b \sqsubset f\underline{a} \sqsubset f\underline{x} \sqsubset f\underline{b} \wedge f b \xrightarrow{\text{ll}} f\underline{b}$		(F$^+$.4d)
$\forall f, w_i, \sigma.$	$w_i = [f]_w \wedge \sigma = [f]_\sigma$	$\implies \sigma_{on} \xrightarrow{\text{rf}} w_i x \sqsubset f \wedge f.\mathsf{B}_{\texttt{cell}} = \sigma.\mathsf{B}[i]$	(F$^+$.5a)
$\forall f \in F_i.$	$\text{def}(f\underline{b})$	$\implies \exists \sigma.\ \sigma = [f]_\sigma \wedge \sigma_{on} \xrightarrow{\text{rf}} f\underline{x}$	(F$^+$.5b)

Fig. 11. Signatures for multi-writer snapshot data-structure with **LL**/**SC** forwarding (Algorithms 2 and 3).

successful forwarding among the concurrent processes, forwarding a value that is current (i.e., written by another overlapping write). This is the key property facilitating linearizability.

Other than that, the SCAN procedure is mostly the same in Algorithm 2 compared to Algorithm 1; the main difference is that the order between s_{on} and each sr_i is swapped, which is done in preparation for Algorithm 3. In Algorithm 1 this swap would result in a bug, causing interference between forwarding and initialization of B; the writes may fail to forward their value because the forwarding flag is turned on too late, or scans may erase forwarded values by the initializations. But this is safe to do in Algorithm 2 because the use of **LL**, **SC** and **VL** over X and B ensures that no forwarding (intended for a scan s' prior to s) can succeed in-between sr_i and s_{on} of s. This is substantiated by the following lemma.

LEMMA 4.4. *Let s' be a scan prior to s, i.e., $s' \sqsubset s$. If $f\underline{x}$ succeeds observing the same s'_{on} that the $w_i x$ preceding $f\underline{x}$ observed, i.e., $s'_{on} \xrightarrow{\text{rf}} f\underline{x}$ and $s'_{on} \xrightarrow{\text{rf}} w_i x$, and $sr_i \lhd f\underline{b}$, then $f\underline{b}$ must fail to write.*

PROOF. We assume that $f\underline{b}$ is successful, and derive contradiction. By Property (M$^+$.1b), there exists some write-like event w such that $w \xrightarrow{\text{rf}} fb$, and by $f\underline{b}$ being successful, Property (M$^+$.7)

implies that $f\underline{b}$ observes the same write-like event as fb, i.e. $w \xrightarrow{\text{rf}} f\underline{b}$. By Property (M$^+$.4), we either have $sr_i \lhd w$ or $w \trianglelefteq sr_i$, in the latter case we have $w \lhd sr_i \lhd f\underline{b}$, contradicting Property (M$^+$.2) by sr_i being in between $w \xrightarrow{\text{rf}} f\underline{b}$, thus we can only have $sr_i \trianglelefteq w$. Since Algorithm 2 is single-scanner, it follows that s only starts after s' has finished, thus we can derive the following:

$$s'_{on} \sqsubset s'_{off} \sqsubset sr_i \trianglelefteq w \lhd fb \lhd f\underline{x}$$

which implies that we have s'_{off} in between $s'_{on} \xrightarrow{\text{rf}} f\underline{x}$, contradicting Property (M$^+$.2). □

4.2.2 The MWForwarding *Signature.* The rep-event signatures for multi-writer algorithms in Fig. 10 correspond closely to the rep events of Algorithm 2. Fig. 10 extends Fig. 6, adding the **LL**/**SC** register X and extending the event signatures. WRITE$_i$+ extends WRITE$_i$ by events w_ix, w_if_1, and w_if_2 from Algorithm 2, along with an event signature FORWARD$_i$ corresponding to the procedure FORWARD from Algorithm 2. VSCAN+ extends VSCAN by updating B to be an array of **LL**/**SC** registers, and adding four additional rep events: σ_{on} and σ_{off} corresponding to s_{on} and s_{off} from Algorithm 2, and $\sigma_{\underline{on}}$ and $\sigma_{\underline{off}}$ which are providing support for multi-scanner Algorithm 3. As we shall see in Section 4.3, in Algorithm 3, σ_{on} might not return before σa_i, and similarly for σ_{off} and σb_i. Thus, we need events that observe σ_{on} and σ_{off} returning before σa_i and σb_i respectively. For Algorithm 2, we simply take $\sigma_{\underline{on}} = \sigma_{on}$ and $\sigma_{\underline{off}} = \sigma_{off}$.

Signature MWForwarding in Fig. 11 captures the common structure of Algorithms 2 and 3. Some aspects are preserved from Algorithm 1; for instance Properties (F.1), (F.2), (F.3a), (F.3b) and (F.4a) are the same as in Forwarding signature of Fig. 7. One difference is that we now have sets of forwarding events F_i that are instances of the forwarding procedure. The new signature introduces operations over such forwardings: $[-]_w$ maps a forwarding to the write that invoked it, and $[-]_\sigma$ maps a forwarding to the virtual scan it is forwarding to. Forwarding visibility $\xrightarrow{\text{fwd}}$ is also updated: it now says that a forwarding reads a value of a write from w_ia and relays this value by $f_i\underline{b}$.

Among the new properties, (F$^+$.3c) encodes that only $f\underline{b}$ performs forwarding writes into σ.B$[i]$. Property (F$^+$.3d) ensures that nothing wrote to X other than σ_{off} directly after σ_{on}. Properties (F$^+$.4b) to (F$^+$.4d) corresponds to the rep event order of scans, writes and forwarding respectively. Lastly, Property (F$^+$.5a) ensures that every forwarding has a virtual scan to forward to, while Property (F$^+$.5b) states that $f\underline{b}$ can only be performed if $f\underline{x}$ observed σ_{on}.

Lemma 4.5. *Histories satisfying* MWForwarding *signature also satisfy* Forwarding *signature.*

Proof sketch. We instantiate Σ and $[-]^\bullet$ of Forwarding to be the same as the corresponding instantiations of MWForwarding. Properties (F.1), (F.2), (F.3a), (F.3b) and (F.4a) are shared between the signatures, therefore they trivially hold.

We next focus on proving Property (F.7b), leaving the remaining properties for Appendix D. Proving Property (F.7b) requires showing that $w_i \xrightarrow{\text{fwd}} \sigma$ and $w'_i \sqsubset \trianglelefteq \xrightarrow{\text{rf}} \sigma$ and $w_i \lhd w'_i$ derive a contradiction. To show this, we need a helper lemma, proved in Appendix D (Lemma D.2): If $w_i \sqsubset \trianglelefteq \xrightarrow{\text{rf}} \sigma$ then surely w_i will be able to forward to σ in time, i.e., we have $\sigma_{off} \not\lhd w_ix$ and $\sigma_{off} \not\lhd f\underline{x}$ and $f\underline{b} \lhd \sigma b_i$ for every f executed by w_i. From $w_i \xrightarrow{\text{fwd}} \sigma$, by the definition of $\xrightarrow{\text{fwd}}$, we have $w_ia \xrightarrow{\text{rf}} f\underline{a}$ and $f\underline{b} \xrightarrow{\text{rf}} \sigma b_i$ for some f. By Lemma 3.1, $w_i \lhd w'_i$ implies $w_ia \lhd w'_ia$. We can also infer (full proof in Appendix D) that $\sigma_{on} \lhd w'_ix$.

By Lemma D.2, we have $\sigma_{off} \not\lhd w_ix$, thus by Property (F$^+$.3d) we derive $\sigma_{on} \xrightarrow{\text{rf}} w_ix$, implying w'_if_1 and w'_if_2 will be executed, let $f' = w'_if_1$ and $f'' = w'_if_2$. By Lemma D.2, we have $\sigma_{off} \not\lhd f'\underline{x}$ and $\sigma_{off} \not\lhd f''\underline{x}$ and $f''\underline{b} \lhd \sigma b_i$, similarly to above with Property (F$^+$.3d) we derive $\sigma_{on} \xrightarrow{\text{rf}} f'\underline{x}$ and $\sigma_{on} \xrightarrow{\text{rf}} f''\underline{x}$, meaning $f'\underline{b}$ and $f''\underline{b}$ will be executed, meaning we have two consecutive **LL**/**SC** pairs of f' and f''. Since we have $\sigma_{on} \lhd w'_ix$ and $f''\underline{b} \lhd \sigma b_i$, it is not possible for any $\sigma' r_i$ to occur in the

intervals of the **LL/SC** pairs, therefore we can apply Lemma 4.3, thus there exists some f''' such that $f'''b \not\lhd f'b$ and $f'''\underline{b} \trianglelefteq f''\underline{b}$.

By Property (M$^+$.4), we either have $f\underline{b} \lhd f'''\underline{b}$ or $f'''\underline{b} \trianglelefteq f\underline{b}$. In the first case, we have

$$f\underline{b} \lhd f'''\underline{b} \trianglelefteq f''\underline{b} \lhd \sigma b_i$$

which contradicts Property (M$^+$.2) by $f'''\underline{b}$ occurring in between $f\underline{b} \xrightarrow{\mathbf{rf}} \sigma b_i$. In the second case, if we have $f'''\underline{b} \lhd f\underline{b}$ then by Lemma 4.1, and $f\underline{b}$ and $f'''\underline{b}$ being successful, we have $f'''\underline{b} \lhd fb$, and this also follows in the case of $f'''\underline{b} = f\underline{b}$. By Property (RB.1) over $w_i'a \sqsubset f'b$ and $f'''b \sqsubset f'''\underline{b}$ we have $w_i'a \sqsubset f'''\underline{b}$ since we cannot have $f'''b \sqsubset f'b$ by $f'''b \not\lhd f'b$. Thus, we have

$$w_ia \lhd w_i'a \sqsubset f'''\underline{b} \lhd fb \sqsubset f\underline{a}$$

which contradicts Property (M.2) by $w_i'a$ occurring in between $w_ia \xrightarrow{\mathbf{rf}} f\underline{a}$. □

Lemma 4.6. *Every execution of Algorithm 2 satisfies the* MWForwarding *signature (Fig. 11).*

Proof. Since this algorithm is single scanner, we can simply define the set of virtual scans to be the same as the set of abs scans and map each abs scan to itself, i.e., $\Sigma = S$ and $s^\bullet = s$. Each rep event directly corresponds to their equivalent variant in Algorithm 2, except for $\sigma_{\underline{on}}$ and $\sigma_{\underline{off}}$, which are set as $\sigma_{\underline{on}} = \sigma_{on} = s_{on}$ and $\sigma_{\underline{off}} = \sigma_{off} = s_{off}$. Property (F.1) holds by each event being a subevent of itself ($e \subseteq e$) and Property (F.4a) holds since the algorithm is single-scanner. Properties (F.2) to (F$^+$.3c) and (F$^+$.4b) to (F$^+$.5b) holds directly by the structure of the algorithm and Property (F$^+$.3d) can be proven by Lemma C.1 from Appendix C, the same lemma that established this property for Algorithm 1, since the relative structure of s_{on} and s_{off} is the same. □

4.3 Multi-Writer, Multi-Scanner

Algorithm 3 is the final of the Jayanti's snapshot algorithms, allowing unconstrained write and scan operations. It differs from Algorithm 2 in the implementation of SCAN, while WRITE is mostly unchanged. While Algorithm 3 allows for multiple concurrently running physical scanners, under the hood, at most one *virtual* scanner can be logically viewed as running at any point in time. The physical scans maintain their own local snapshot and forwarding arrays, but the shared resource X, via the fields X.p$_\mathsf{A}$ and X.p$_\mathsf{B}$, keeps track which local snapshot array and local forwarding array, respectively, should be considered as the current data of the virtual scan. The physical scanners race to modify these fields in order to promote their own local arrays as the arrays of the virtual scan. However, X can be modified only by **SC**; thus, by Lemma 4.1, mutations to X occur sequentially, and by Lemma 4.2, at least one mutation exists. The sequential nature of these mutations justifies viewing them as belonging to a single, albeit virtual, scan. Once the virtual scan terminates, a snapshot is copied into SS ($v_{\underline{ss}}$), which scanners read (s_{ss}) and return. We follow Jayanti in assuming that SS, even though physically a compound object, can still be considered an **LL/SC** register; this is not a loss of generality, as multi-word **LL/SC** registers exist [Jayanti and Petrovic 2005].

The PUSHVS procedure (short for "push virtual scan") propels the virtual scanner to its completion, producing a snapshot in SS. The procedure is divided into three phases, tracked by the field X.phase, corresponding to the three phases of the previous algorithms: the first phase sets each memory cell in B to $\perp$ with vr_i; the second phase reads the main memory A with va_i; and the third phase reads the forwarding memory B with vb_i. However, unlike before when B was a global array, here we have one B$_p$ for each scan process p. A scanner that successfully writes with v_{on} in the first phase will have its B$_p$ used for forwarding in the current virtual scan. For the second phase, the algorithm uses A$_p$ to store the results of reading A, and similarly the process p which successfully writes with v_{off} will have its A$_p$ used for the next phase. The third phase uses A$_p$ and B$_q$ from the processes stored in X to construct the snapshot, which is then stored in SS with $v_{\underline{ss}}$.

resource $\mathsf{A} : \mathbf{array}[n]\ \mathbf{of}\ \mathsf{val}$
resource $\mathsf{A}_p : \mathbf{array}[n]\ \mathbf{of}\ \mathsf{val}$
resource $\mathsf{B}_p : \mathbf{array}[n]\ \mathbf{of}\ \mathsf{val} \cup \{\bot\}$
resource $\mathsf{X} : \left\langle \begin{array}{l} \mathsf{phase} : \{1,2,3\} \coloneqq 1 \\ \mathsf{p_A}\ \mathsf{p_B} : \mathsf{PID} \\ \mathsf{sync} : \mathbb{B} \coloneqq \mathtt{false} \end{array} \right\rangle$
resource $\mathsf{SS} : \left\langle \begin{array}{l} \mathsf{img} : \mathbf{array}[n]\ \mathbf{of}\ \mathsf{val} \\ \mathsf{sync} : \mathbb{B} \coloneqq \mathtt{false} \end{array} \right\rangle$

1: WRITE($i : \mathbb{N}, v : \mathsf{val}$) $\triangleq$ ▷ w_i
2: $\mathsf{A}[i] \coloneqq v$ ▷ $w_i a$
3: $x \leftarrow \mathbf{LL}(\mathsf{X})$ ▷ $w_i x$
4: **if** $x.\mathsf{phase} = 2$ **then**
5: FORWARD($i, x.\mathsf{p_B}$) ▷ $w_i f_1$
6: FORWARD($i, x.\mathsf{p_B}$) ▷ $w_i f_2$

7: FORWARD($i : \mathbb{N}, p : \mathsf{PID}$) $\triangleq$ ▷ f
8: $\mathbf{LL}(\mathsf{B}_p[i])$ ▷ fb
9: $v \leftarrow \mathsf{A}[i]$ ▷ $f\underline{a}$
10: $x' \leftarrow \mathbf{VL}(\mathsf{X})$ ▷ $f\underline{x}$
11: **if** x' **then** $\mathbf{SC}(\mathsf{B}_p[i], v)$ ▷ $f\underline{b}$

12: SCAN: $\mathbf{array}[n]\ \mathbf{of}\ \mathsf{val}$ $\triangleq$ ▷ s
13: $p \leftarrow \mathtt{getPID}$
14: PUSHVS(p) ▷ sv_1
15: PUSHVS(p) ▷ sv_2
16: $ss \leftarrow \mathsf{SS}$ ▷ s_{ss}
17: **return** $ss.\mathsf{img}$

18: PUSHVS($p : \mathsf{PID}$) $\triangleq$ ▷ v
19: $x \leftarrow \mathbf{LL}(\mathsf{X})$ ▷ vx
20: **if** $x.\mathsf{phase} = 1$ **then**
21: **for** $i \in \{0 \ldots n-1\}$ **do**
22: $\mathsf{B}_p[i] \coloneqq \bot$ ▷ vr_i
23: $\mathbf{SC}(\mathsf{X}, \langle 2, x.\mathsf{p_A}, p, x.\mathsf{sync}\rangle)$ ▷ v_{on}
24: $x \leftarrow \mathbf{LL}(\mathsf{X})$ ▷ vx
25: **if** $x.\mathsf{phase} = 2$ **then**
26: **for** $i \in \{0 \ldots n-1\}$ **do**
27: $a \leftarrow \mathsf{A}[i]$ ▷ va_i
28: $\mathsf{A}_p[i] \coloneqq a$ ▷ $v\bar{a}_i$
29: $\mathbf{SC}(\mathsf{X}, \langle 3, p, x.\mathsf{p_B}, x.\mathsf{sync}\rangle)$ ▷ v_{off}
30: $x \leftarrow \mathbf{LL}(\mathsf{X})$ ▷ vx
31: **if** $x.\mathsf{phase} = 3$ **then**
32: **for** $i \in \{0 \ldots n-1\}$ **do**
33: $b \leftarrow \mathsf{B}_{x.\mathsf{p_B}}[i]$ ▷ vb_i
34: **if** $b \neq \bot$ **then**
35: $V[i] \coloneqq b$
36: **else**
37: $a \leftarrow \mathsf{A}_{x.\mathsf{p_A}}[i]$ ▷ $v\underline{a}_i$
38: $V[i] \coloneqq a$
39: $ss \leftarrow \mathbf{LL}(\mathsf{SS})$ ▷ v_{ss}
40: $x' \leftarrow \mathbf{VL}(\mathsf{X})$ ▷ $v\underline{x}$
41: **if** $ss.\mathsf{sync} = x.\mathsf{sync} \wedge x'$ **then**
42: $\mathbf{SC}(\mathsf{SS}, \langle V, \neg ss.\mathsf{sync}\rangle)$ ▷ $v_{\underline{ss}}$
43: $\mathbf{SC}(\mathsf{X}, \langle 1, x.\mathsf{p_A}, x.\mathsf{p_B}, \neg x.\mathsf{sync}\rangle)$ ▷ v_{end}

Algorithm 3. Jayanti's multi-writer, multi-scanner snapshot algorithm.

Each phase starts with an $\mathbf{LL}(\mathsf{X})$ (three lines labeled vx) and ends with an $\mathbf{SC}(\mathsf{X})$. This ensures that if multiple physical scanners run a phase simultaneously, only a single SC operation succeeds writing to X, committing the results of the phase. Thus, considered across all physical scanners, successful phases cannot overlap, by Lemma 4.1. Also, phases must run consecutively, because a phase in some scanner can only start if the previous phase, maybe executed by some other scanner, has been terminated, incrementing X.phase to enable the next phase (v_{on}, v_{off}, and v_{end}). Because the phases cannot overlap, must be executed in order, and correspond to the scanner phases in the previous algorithms, a trace of a virtual scan in Algorithm 3 has essentially the same structure as a trace of a physical scan in the Algorithms 1 and 2.

To ensure that SS is only mutated once per virtual scan, we have two toggle bits X.sync and SS.sync as part of X and SS respectively. These function such that every time v_{end} successfully writes, X.sync is negated. SS can only be written to by $v_{\underline{ss}}$, which can only be executed if X.sync and SS.sync are equal[2] and X did not change during the phase. Once $v_{\underline{ss}}$ successfully writes, SS.sync will be negated, meaning no more writes into SS can occur in this phase since X.sync and SS.sync are now distinct. We also know that there has to be exactly one successful write with $v_{\underline{ss}}$ before v_{end}, since $v_{\underline{ss}}$ only fails if some other $v_{\underline{ss}}$ succeeded, or some other v_{end} ended the phase.

[2]Jayanti's original presentation is dual, as the sync bits need to be distinct for writing to SS, but this is equivalent.

It is also vital for linearizability that the virtual scan is in the time interval of the abs SCAN returning it. To ensure the virtual scan ends before the abs scanner ends, the latest virtual scan is finished before PUSHVS terminates. Dually, to ensure that the virtual scan was not started before the abs scanner starts, PUSHVS is executed twice; once to finish the currently running virtual scan, and once to generate a virtual scan that can be used by the abs scanner. Executing PUSHVS twice is similar to the idea of executing FORWARD twice in Algorithm 2.

LEMMA 4.7. *Every execution of Algorithm 3 satisfies the* MWForwarding *signature (Fig. 11).*

PROOF SKETCH. We instantiate the write and forward rep events for the event structures of Fig. 10 with rep events of the same name. The structure of WRITE and FORWARD ensures that Properties (F.3a), (F⁺.3c) and (F⁺.4c) to (F⁺.5b) are satisfied.

We instantiate virtual scans as a logical object corresponding to the rep events executed by PUSHVS constructing a complete virtual scan. The interval of a virtual scan σ is instantiated as the smallest interval containing all its rep events. More formally, if we have v, v' and v'' such that $v_{on} \xrightarrow{\text{rf}} vx'$ and $v'_{off} \xrightarrow{\text{rf}} vx''$ and $v''_{\underline{ss}}$ wrote to SS, then there exists a virtual scan σ. Its rep events are instantiated using the rep events of v, v' and v'' as follows and instantiate $s^\bullet = \sigma$ iff $v''_{\underline{ss}} \xrightarrow{\text{rf}} s_{ss}$.

$$\sigma r_i \triangleq vr_i \qquad \sigma_{on} \triangleq v_{on} \qquad \sigma_{\underline{on}} \triangleq vx' \qquad \sigma a_i \triangleq va'_i \qquad \sigma_{off} \triangleq v'_{off} \qquad \sigma_{\underline{off}} \triangleq vx'' \qquad \sigma b_i \triangleq vb''_i$$

We also let $\sigma_{init} \triangleq vx$ and $\sigma_{\underline{ss}} \triangleq v''_{\underline{ss}}$ and $\sigma_{end} \triangleq v'''_{end}$ where v'''_{end} was successful and $v'_{off} \xrightarrow{\text{rf}} v'''_{end}$, however these do not need to be in the interval of σ. Properties (F.2), (F.3b), (F⁺.3d) and (F⁺.4b) follows from this structure, which we show in Appendix E.

The most involved part is proving Properties (F.1) and (F.4a), corresponding to $s^\bullet \subseteq s$ and $(\sigma \sqsubset \sigma' \vee \sigma' \sqsubset \sigma)$ respectively, which require us to establish the following helper lemmas, presented in Appendix E: By Lemma E.2, we have exactly one $\sigma_{\underline{ss}}$ per σ, satisfying $\sigma_{\underline{ss}} \lhd \sigma_{end}$, and by Lemma E.5, for any terminated v and v' of PUSHVS with $v \sqsubset v'$, there exists σ and σ' such that $\sigma_{end} \lhd \sigma'_{end}$ and $vx \not\lhd \sigma_{end}$. With Lemma E.2, we can that show for every i we either have $\sigma b_i \sqsubset \sigma_{\underline{ss}} \lhd \sigma_{end} \lhd \sigma'_{init} \sqsubset \sigma' r_i$ or $\sigma' b_i \sqsubset \sigma'_{\underline{ss}} \lhd \sigma'_{end} \lhd \sigma_{init} \sqsubset \sigma r_i$, implying Property (F.4a). With Lemma E.5, we can show for s that there exists some σ' and σ'' with $\sigma'_{end} \lhd \sigma''_{end}$ and $vx \not\lhd \sigma'_{end}$ where $vx \subseteq sv_1$. This implies that if we have $\sigma = s^\bullet$, then $\sigma'_{end} \lhd \sigma_{init}$, since some new virtual scan must have started after σ'_{end}, and by the instantiation of $[-]^\bullet$ we have $\sigma_{\underline{ss}} \xrightarrow{\text{rf}} s_{ss}$, thus for all i we have $\sigma'_{end} \lhd \sigma_{init} \sqsubset \sigma r_i \sqsubset \sigma b_i \sqsubset \sigma_{\underline{ss}} \xrightarrow{\text{rf}} s_{ss}$, implying $\sigma \subseteq s$ and Property (F.1), the last step is further explained in Appendix E. □

5 APPLYING TO OTHER SNAPSHOT ALGORITHMS

To show the generality of our approach, we next apply it to the snapshot algorithm of Afek et al. [1993], presented as Algorithm 4. It is a single-writer, multi-scanner algorithm. At its core, SCAN of Algorithm 4 collects snapshots by reading the memory array A twice. We denote the first read of index i by $s_k a_i$, and the second by $s_k b_i$. The algorithm looks for changes in the array. If a change is detected, the reading is restarted in the next iteration. The index k on the reading events identifies the iteration in which the event occurs. If no change to the array is detected between the first and second read events, then what was read is a valid snapshot, reflecting what was in the array at the moment the last index was first read in the current iteration. To ensure that changes to the array are properly recognized, each memory cell holds a version number (`A[i].ver`) of the latest write, which is a number that is incremented each time a new value is written to the cell.

If SCAN detects that some index i has changed twice during scanning, it can immediately terminate, returning the *view* of the latest write into i (`A[i].view`). The view itself is a snapshot each WRITE collects by calling SCAN before writing its value. Keeping views ensures that the snapshot methods are wait-free. SCAN tracks how many times index i has changed by updating the local

```
resource A : array[n] of ⟨ data : val, ver : ℕ, view : array[n] of val ⟩

1: WRITE(i : ℕ, v : val) ≜                                   ▷ w_i
2:     s := SCAN()                                           ▷ w_i s
3:     A[i] := ⟨v, A[i].ver + 1, s⟩                          ▷ w_i a

4: SCAN: array[n] of val ≜                                   ▷ s
5:     for i ∈ {0 ... n − 1} do
6:         moved[i] := false
7:     while true do                          k-th iteration of loop
8:         for i ∈ {0 ... n − 1} do
9:             a[i] := A[i]                                  ▷ s_k a_i
10:        for i ∈ {0 ... n − 1} do
11:            b[i] := A[i]                                  ▷ s_k b_i
12:        changed := false
13:        for i ∈ {0 ... n − 1} do
14:            if a[i].ver ≠ b[i].ver then
15:                if moved[i] then
16:                    return b[i].view
17:                else
18:                    changed := true
19:                    moved[i] := true
20:        if ¬changed then
21:            return (b[0].data, ..., b[n − 1].data)
22:    end while
```

Algorithm 4. Single-writer, multi-scanner snapshot algorithm of Afek et al. [1993].

variable $moved[i]$ each time it detects a change. Each WRITE commits its value simultaneously with incrementing the version number and updating the view.

We next use visibility relations to show that Algorithm 4 satisfies the Snapshot signature, and is thus linearizable by Lemma 2.3.

LEMMA 5.1. *Every execution of Algorithm 4 satisfies the* Snapshot *signature (Fig. 5).*

PROOF SKETCH. To prove that Algorithm 4 satisfies the Snapshot signature, we employ virtual scans as a simplification mechanism that allows us to elide WRITE views from consideration when establishing Snapshot. A virtual scan will denote a scan that detected no change in the array, i.e., it returned at line 21. To each abs scan, we can associate a virtual scan as follows. If the abs scan terminated because it detected no change, than it immediately is a virtual scan. If the abs scan terminated by returning a view from a WRITE, i.e., returned at line 16, then that view itself is an abs scan which can, recursively, be associated with a virtual scan. Importantly, given abs scan s, the associated virtual scan $s^\bullet$ satisfies $s^\bullet \subseteq s$, from which we shall derive the Snapshot axioms.

Formally, a virtual scan σ consists of rep events σa_i and σb_i for each i, where both σa_i and σb_i observe the same write (i.e., no change detected), and every σa_i occurs before any σb_j:

$$\forall \sigma, i. \qquad \exists w_i.\ w_i a \xrightarrow{\text{rf}} \sigma a_i \wedge w_i a \xrightarrow{\text{rf}} \sigma b_i \tag{A.1}$$

$$\forall \sigma, i, j. \qquad \sigma a_i \sqsubset \sigma b_j \tag{A.2}$$

If scan s returned at line 21, we simply define the virtual scan $s^\bullet$ as the set of rep events $s_k a_i$ and $s_k b_i$, where k is the last reading iteration of s. If scan s returned at line 16, then s returns the view of some w_i, and we define $s^\bullet = (w_i s)^\bullet$. This is a well-founded recursive definition, because the ending time of the scan $w_i s$ on the right is smaller than the ending time of s on the left, and the ending times are bounded from below by 0. To see that the ending time of $w_i s$ is below that of s, suppose otherwise. Then it must be $s \sqsubset w_i a$, because from the code of WRITE we have that $w_i s \sqsubset w_i a$. In particular, $s_k b_i \sqsubset w_i a$ for every k. But, because $w_i a$ is observed by some rep read in s, we also have $w_i a \xrightarrow{\text{rf}} s_k b_i$ for some k. Thus, we have an event $s_k b_i$ that terminated before the event $w_i a$ that it observes, which contradicts Property (V.1).

Next, we show that $s^\bullet \subseteq s$. If s terminated with no changes detected, this is trivial, since $s^\bullet$ consists of the selected rep events of s. Otherwise, $s^\bullet = (w_i s)^\bullet$, for some write view $w_i s$. By recursion on the definition of $[-]^\bullet$ it must be $s^\bullet = w_i s^\bullet \subseteq w_i s$, so it suffices to show $w_i s \subseteq s$. This holds

because, to reach the return at line 16, two changes of $\mathtt{A}[i]$ must have occurred, thus there are two different writes to $\mathtt{A}[i]$ before w_i that where observed by some rep event in s, implying s starts before w_i and thus also before $w_i s$. Additionally, since some $s_k b_i$ must have read $w_i a$, i.e., $w_i a \xrightarrow{\mathrm{rf}} s_k b_i$, it must be that $w_i s$ ends before s, as otherwise $s_k b_i \sqsubset w_i a$ contradicts Property (V.1).

We next proceed to establish the Snapshot signature. First, we instantiate $\mathbb{W}_i$ to be the set of all writes $w_i \in W_i$ where $w_i a$ is defined; these are the writes that executed their effect. Second, we instantiate the visibility relations $\prec$ and $\xrightarrow{\mathrm{rf}}$ as follows, using the helper relation $\xrightarrow{\mathrm{rf}}$.

$$w_i \xrightarrow{\mathrm{rf}} \sigma \triangleq w_i a \xrightarrow{\mathrm{rf}} \sigma a_i \qquad w_i \xrightarrow{\mathrm{rf}} s \triangleq w_i \xrightarrow{\mathrm{rf}} s^\bullet \qquad \prec \triangleq \xrightarrow{\mathrm{rf}}$$

In English, the scan s reads from, and also observes, w_i iff there is an appropriate rep event in the virtual scan $s^\bullet$ that reads from $w_i a$ at the level of rep events. We now argue that the definitions of the visibility relations satisfy the Snapshot properties. Property (V.1) holds since if $e \prec^+ e'$ and $e' \sqsubseteq e$ then $e \prec^+ e'$ can only hold if $e \xrightarrow{\mathrm{rf}} e'$, i.e., $w_i a \xrightarrow{\mathrm{rf}} \sigma a_i$. But then, by Property (RB.2), from $e' \sqsubseteq e$ we derive $\sigma a_i \sqsubset w_i a$, which contradicts Property (V.1) for registers. Property (S.1) holds by the structure of the algorithm. Property (S.3) holds since if $w_i \xrightarrow{\mathrm{rf}} s$ and $w_i' \xrightarrow{\mathrm{rf}} s$, then $w_i a$ and $w_i' a$ are observed by the same read. Thus, by Property (M.3), $w_i a = w_i' a$, and since each write executes $w_i a$ only once by Property (ES.2), it must be $w_i = w_i'$. Property (S.4) holds because the algorithm is assumed to be single-writer. Property (S.5) holds since each terminated write must have $w_i a$ defined, which is how we define the set $\mathbb{W}_i$.

Next, Property (S.2) says that given $w_i \xrightarrow{\mathrm{rf}} s$, there exists no w_i' such that $w_i \lhd w_i' \lhd s$. To prove it, we assume that w_i' exists, and derive a contradiction. By assumption, we have $w_i \xrightarrow{\mathrm{rf}} \sigma$ for $\sigma = s^\bullet$. From $\sigma = s^\bullet \subseteq s$, and $w_i \lhd w_i' \lhd s$, we can derive $w_i \lhd w_i' \lhd \sigma$, and then by Property (A.2), also $w_i a \lhd w_i' a \lhd \sigma b_i$. We elide the proof of the last derivation; it is similar to Lemma 3.1 and is in Appendix F. By the definition of $\xrightarrow{\mathrm{rf}}$ and Property (A.1), $w_i \xrightarrow{\mathrm{rf}} \sigma$ implies $w_i a \xrightarrow{\mathrm{rf}} \sigma a_i, \sigma b_i$. But then $w_i' a$ occurrs between $w_i a$ and σb_i, contradicting Property (M.2) and $w_i a \xrightarrow{\mathrm{rf}} \sigma b_i$.

Lastly, we prove Property (S.6): $w_i, w_j \xrightarrow{\mathrm{rf}} s$ and $w_i', w_j' \xrightarrow{\mathrm{rf}} s'$ with $w_i \lhd w_i'$ and $w_j' \lhd w_j$ lead to contradiction. We unfold the assumptions $w_i, w_j \xrightarrow{\mathrm{rf}} s$ and $w_i', w_j' \xrightarrow{\mathrm{rf}} s'$ into $w_i, w_j \xrightarrow{\mathrm{rf}} \sigma$ and $w_i', w_j' \xrightarrow{\mathrm{rf}} \sigma'$, where $\sigma = s^\bullet$ and $\sigma' = s'^\bullet$. By definition of $\xrightarrow{\mathrm{rf}}$ and Property (A.1), from $w_j \xrightarrow{\mathrm{rf}} \sigma$ we have $w_j a \xrightarrow{\mathrm{rf}} \sigma a_j, \sigma b_j$, and similarly for $w_j' \xrightarrow{\mathrm{rf}} \sigma'$. By Property (A.2) we have $\sigma a_j \sqsubset \sigma b_i$ and $\sigma' a_i \sqsubset \sigma' b_j$, and by Property (RB.1), it is either $\sigma a_j \sqsubset \sigma' b_j$ or $\sigma' a_i \sqsubset \sigma b_i$. In the first case (the second is symmetric) we can construct $w_j' a \lhd w_j a \xrightarrow{\mathrm{rf}} \sigma a_j \sqsubset \sigma' b_j$, meaning that we have $w_j' a \lhd w_j a \lhd \sigma' b_j$ contradicting Property (M.2) for $w_j' a \xrightarrow{\mathrm{rf}} \sigma' b_j$ by $w_j a$ occurring between $w_j' a$ and $\sigma' b_j$. □

We conclude this section by observing that the inverse of Lemma 2.3 also holds; that is, Snapshot signature is actually satisfiable by every linearizable snapshot algorithm. This follows by instantiating $\prec$ with the linearization order, defining $\xrightarrow{\mathrm{rf}}$ to relate each read with the latest write before that read in the linearization, and defining $\mathbb{W}_i$ as the set of all writes in the linearization. The axioms of the signature then simply state straightforward properties of the linearization order and of sequential execution in that order. That said, we have established Snapshot signature for Algorithm 4 *without* assuming linearizability, so that we can derive linearizability by Lemma 2.3.

6 RELATED WORK

The idea to use visibility relations for proving linearizability has first been proposed by Henzinger et al. [2013] and applied to concurrent queue algorithms. The explicit motivation of this approach was to modularize the linearizability proofs. The work on time-stamped stack algorithm of Dodds et al. [2015] builds on this by introducing a hybrid approach, combining the visibility method together with the linearization points method. Visibility has also been applied on algorithms over weak memory, where it is typically called communication order [Raad et al. 2019]. Another

related approach of visibility by Emmi and Enea [2019]; Krishna et al. [2020] formulates weakened specifications for linearizability, where all operations do not have to linearly occur one after another in a simulation, but occur in partial order, allowing for certain operations to miss another.

Concerning Jayanti's algorithms specifically, Jayanti [2005] sketched linearizability of the first algorithm by describing its linearization points. He also argued informally, based on forwarding principles, that the changes between the algorithms preserve linearizability. Delbianco et al. [2017] gave a proof formalized in Coq of the first algorithm, by constructing a linearization order along the execution of the algorithm. The proof is specific to the implementation, making it is unclear how to lift it to the other two algorithms. Based on that development, Jacobs [2018] developed a mechanized proof, also of the first algorithm, in VeriFast using prophecy variables. Petrank and Timnat [2013] and Timnat [2015] present an algorithm based on the forwarding idea of Jayanti, which implements a set interface with insert, remove and contains operations, along with an iterator which is a generalized form of scanner. The set operations can be forwarded to the scanner, generalizing Jayanti's forwarding which applies only to write operations. In the future, we will consider how to generalize our proof to apply to this algorithm as well. This would involve axiomatizing the set data-structure, but also what it means abstractly to be a forwarding structure, so that both Jayanti's snapshot and the set structure of Petrank and Timnat are instances.

Afek et al. [1993] proved their algorithm linearizable by using the linearization point method. While the use of the linearization point method is fairly straightforward for this algorithm, we showed that we can reuse our snapshot axiomatization for its proof.

7 CONCLUSION AND FUTURE WORK

We presented proofs of linearizability of Jayanti's three snapshot algorithms developed in a modular fashion, using the method of visibility relations. More concretely, the linearizability proofs are decomposed into proof modules, with many of the modules being shared between the three algorithms; they are developed once and reused three times to reduce proof complexity.

Importantly, the module interfaces are signatures consisting of relations and axioms on them that encode the key idea of "forwarding principles" underpinning Jayanti's design. We thus show that a formalism based on visibility relations is powerful enough to mathematically capture these principles; previously, Jayanti has only presented them informally in English.

In the future, we plan to apply the visibility methods to other algorithms, and potentially also use the developed modules and signatures as guides in designing new and more efficient algorithms, much like Jayanti's three algorithms start with the single-writer/single-scanner variant and build to the ultimately desired multi-writer/multi-scanner variant. We shall also study how the visibility method applies to non-linearizable algorithms, and how to mechanize our proof in a proof assistant.

ACKNOWLEDGMENTS

We thank the anonymous reviewers from the POPL'22 PC for their feedback. This research was partially supported by the Spanish MICINN project BOSCO (PGC2018-102210-B-I00) and the European Research Council project Mathador (ERC2016-COG-724464).

REFERENCES

Martín Abadi and Leslie Lamport. 1991. The existence of refinement mappings. *Theoretical Computer Science* 82, 2 (1991), 253–284. https://doi.org/10.1016/0304-3975(91)90224-P

Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. 1993. Atomic snapshots of shared memory. *J. ACM* 40, 4 (Sept. 1993), 873–890. https://doi.org/10.1145/153724.153741

Ahmed Bouajjani, Michael Emmi, Constantin Enea, and Suha Orhun Mutluergil. 2017. Proving linearizability using forward simulations. In *Computer-Aided Verification (CAV) (LNCS, Vol. 10427)*. 542–563. https://doi.org/10.1007/978-3-319-63390-9_28

Soham Chakraborty, Thomas A. Henzinger, Ali Sezgin, and Viktor Vafeiadis. 2015. Aspect-oriented linearizability proofs. *Logical Methods in Computer Science* Volume 11, Issue 1 (2015). https://doi.org/10.2168/LMCS-11(1:20)2015

Germán Andrés Delbianco, Ilya Sergey, Aleksandar Nanevski, and Anindya Banerjee. 2017. Concurrent Data Structures Linked in Time. In *European Conference on Object-Oriented Programming, ECOOP (LIPIcs, Vol. 74)*. 8:1–8:30. https://doi.org/10.4230/LIPIcs.ECOOP.2017.8

Mike Dodds, Andreas Haas, and Christoph M. Kirsch. 2015. A scalable, correct time-stamped stack. In *Symposium on Principles of Programming Languages (POPL)*. 233–246. https://doi.org/10.1145/2676726.2676963

Michael Emmi and Constantin Enea. 2019. Weak-consistency specification via visibility relaxation. *Proc. ACM Program. Lang.* 3, POPL (2019), 60:1–60:28. https://doi.org/10.1145/3290373

Stefan Felsner. 1992. *Interval orders: combinatorial structure and algorithms.* Ph.D. Dissertation. Technical University of Berlin. http://page.math.tu-berlin.de/~felsner/Paper/diss.pdf

Thomas A. Henzinger, Ali Sezgin, and Viktor Vafeiadis. 2013. Aspect-oriented linearizability proofs. In *International Conference on Concurrency Theory (CONCUR)*. 242–256. https://doi.org/10.1007/978-3-642-40184-8_18

Maurice Herlihy and Nir Shavit. 2008. *The art of multiprocessor programming.* M. Kaufmann. https://doi.org/10.1108/03684920810907904

Maurice P. Herlihy and Jeannette M. Wing. 1990. Linearizability: a correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.* 12, 3 (1990), 463–492. https://doi.org/10.1145/78969.78972

Bart Jacobs. 2018. Jayanti's algorithm using prophecies in VeriFast. https://github.com/verifast/verifast/blob/master/examples/jayanti

Bart Jacobs, Willem Penninckx, and Amin Timany. 2018. *Abstract I/O specification.* Technical report CW714. Department Of Computer Science KU Leuven. https://lirias.kuleuven.be/2088062

Prasad Jayanti. 2005. An optimal multi-writer snapshot algorithm. In *Symposium on Theory of Computing (STOC '05)*. 723–732. https://doi.org/10.1145/1060590.1060697

Prasad Jayanti and Srdjan Petrovic. 2005. Efficient wait-free implementation of multiword LL/SC variables. In *International Conference on Distributed Computing Systems (ICDCS)*. 59–68. https://doi.org/10.1109/ICDCS.2005.29

Eric H Jensen, Gary W Hagensen, and Jeffrey M Broughton. 1987. *A new approach to exclusive data access in shared memory multiprocessors.* Technical report UCRL-97663. Lawrence Livermore National Laboratory. https://llnl.primo.exlibrisgroup.com/permalink/01LLNL_INST/1g1o79t/alma991001081569706316

Ralf Jung, Rodolphe Lepigre, Gaurav Parthasarathy, Marianna Rapoport, Amin Timany, Derek Dreyer, and Bart Jacobs. 2020. The future is ours: prophecy variables in separation logic. *Proc. ACM Program. Lang.* 4, POPL (2020), 45:1–45:32. https://doi.org/10.1145/3371113

Siddharth Krishna, Michael Emmi, Constantin Enea, and Dejan Jovanovic. 2020. Verifying visibility-based weak consistency. In *European Symposium on Programming (ESOP) (LNCS, Vol. 12075)*. 280–307. https://doi.org/10.1007/978-3-030-44914-8_11

Nancy A. Lynch and Frits W. Vaandrager. 1995. Forward and Backward Simulations: I. Untimed Systems. *Inf. Comput.* 121, 2 (1995), 214–233. https://doi.org/10.1006/inco.1995.1134

Erez Petrank and Shahar Timnat. 2013. Lock-free data-structure iterators. In *International Symposium on Distributed Computing (DISC)*. 224–238. https://doi.org/10.1007/978-3-642-41527-2_16

Azalea Raad, Marko Doko, Lovro Rožić, Ori Lahav, and Viktor Vafeiadis. 2019. On library correctness under weak memory consistency: specifying and verifying concurrent libraries under declarative consistency models. *Proc. ACM Program. Lang.* 3, POPL, Article 68 (2019). https://doi.org/10.1145/3290381

Shahar Timnat. 2015. *Practical parallel data structures.* Ph.D. Dissertation. Computer Science Department, Technion. http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi/2015/PHD/PHD-2015-06

Paolo Viotti and Marko Vukolić. 2016. Consistency in non-transactional distributed storage systems. *ACM Comput. Surv.* 49, 1, Article 19 (2016). https://doi.org/10.1145/2926965