Transaction Chopping for Parallel Snapshot Isolation

Andrea Cerone[1], Alexey Gotsman[1], and Hongseok Yang[2]

[1] IMDEA Software Institute
[2] University of Oxford

Abstract. Modern Internet services often achieve scalability and availability by relying on large-scale distributed databases that provide consistency models for transactions weaker than serialisability. We investigate the classical problem of transaction chopping for a promising consistency model in this class—parallel snapshot isolation (PSI), which weakens the classical snapshot isolation to allow more efficient large-scale implementations. Namely, we propose a criterion for checking when a set of transactions executing on PSI can be chopped into smaller pieces without introducing new behaviours, thus improving efficiency. We find that our criterion is more permissive than the existing one for chopping serialisable transactions. To establish our criterion, we propose a novel declarative specification of PSI that does not refer to implementation-level concepts and, thus, allows reasoning about the behaviour of PSI databases more easily. Our results contribute to building a theory of consistency models for modern large-scale databases.

1 Introduction

Modern Internet services often achieve scalability and availability by relying on databases that replicate data across a large number of nodes and/or a wide geographical span [17, 21, 24]. The database clients can execute transactions on the data at any of the replicas, which communicate changes to each other using message passing. Ideally, we want this distributed system to provide strong guarantees about transaction processing, such as serialisability [9]. Unfortunately, achieving this requires excessive synchronisation among replicas, which increases latency and limits scalability [1, 14]. For this reason, modern large-scale databases often provide weaker consistency models that allow non-serialisable behaviours, called *anomalies*. Recent years have seen a plethora of consistency model proposals that make different trade-offs between consistency and performance [6, 7, 19, 21]. Unfortunately, whereas transactional consistency models have been well-studied in the settings of smaller-scale databases [2, 12, 20] and transactional memory [5, 11, 13, 15], models for large-scale distributed databases are poorly understood. In particular, we currently lack a rich theory that would guide programmers in using such models correctly and efficiently.

In this paper we make a step towards building such a theory by investigating the classical problem of transaction chopping [20] for a promising consistency model of *parallel snapshot isolation (PSI)* [21]. PSI weakens the classical *snapshot isolation (SI)* [8] in a way that allows more efficient large-scale implementations. Like in SI, a transaction in PSI reads values of objects in the database from a snapshot taken at its start. Like SI, PSI precludes *write conflicts*: when two concurrent transactions write

(a) Original transactions.

```
txn lookup(acct) {
 return acct.balance; }

txn transfer(acct1,acct2,amnt) {
 acct1.balance -= amnt;
 acct2.balance += amnt; }
```

(b) A chopping of `transfer` (`lookup` is left as is).

```
txn withdraw(acct,amnt) {
 acct.balance -= amnt; }

txn deposit(acct,amnt) {
 acct.balance += amnt; }

chain transfer(acct1,acct2,amnt)
{ withdraw(acct1,amnt);
  deposit(acct2,amnt); }
```

(c) An additional transaction making the chopping incorrect.

```
txn lookup2(acct1,acct2) {
 return acct1.balance+acct2.balance }
```

Fig. 1. Example of chopping transactions.

to the same object, one of them must abort. A PSI transaction initially commits at a single replica, after which its effects are propagated asynchronously to other replicas. Unlike SI, PSI does not enforce a global ordering on committed transactions: these are propagated between replicas in *causal* order. This ensures that, if Alice posts a message that is seen by Bob, and Bob posts a response, no user can see Bob's response without also seeing Alice's original post. However, causal propagation allows two clients to see concurrent events as occurring in different orders: if Alice and Bob concurrently post messages, then Carol may initially see Alice's message, but not Bob's, and Dave may see Bob's message, but not Alice's.

A common guideline for programmers using relational databases and transactional memory is to keep transactions as short as possible to maximise performance; long transactions should be *chopped* into smaller pieces [3, 20, 23]. This advice is also applicable to PSI databases: the longer a transaction, the higher the chances that it will abort due to a write conflict. Unfortunately, the subtle semantics of PSI makes it non-trivial to see when a transaction can be chopped without introducing undesirable behaviours. In this paper, we determine conditions that ensure this. In more detail, we assume that the code of all transactions operating on the database is known. As a toy example, consider the transactions in Figure 1(a), which allow looking up the balance of an account `acct` and transferring an amount `amnt` from an account `acct1` to an account `acct2` (with a possibility of an overdraft). To improve the efficiency of `transfer`, we may chop this transaction into a *chain* [24] of smaller transactions in Figure 1(b), which the database will execute in the order given: a `withdraw` transaction on the account `acct1` and a `deposit` transaction on the account `acct2`. This chopping is *correct* in that any client-observable behaviour of the resulting chains could be produced by the original unchopped transactions. Intuitively, even though the chopping in Figure 1(b) allows a database state where `amnt` is missing from both accounts, a client cannot notice this, because it can only query the balance of a single account. If we added the transaction `lookup2` in Figure 1(c), which returns the sum of the accounts `acct1` and `acct2`, then

the chopping of `transfer` would become incorrect: by executing `lookup2` a client could observe the state with `amnt` missing from both accounts.

We propose a criterion that ensures the correctness of a given chopping of transactions executing on PSI (§5). Our criterion is weaker than the existing criterion for chopping serialisable transactions by Shasha et al. [20]: weakening consistency allows more flexibility in optimising transactions. Recent work has shown that transactions arising in web applications can be chopped in a way that drastically improves their performance when executed in serialisable databases [18, 24]. Our result enables bringing these benefits to databases providing PSI.

A challenge we have to deal with in proposing a criterion for transaction chopping is that the specification of PSI [21] is given in a low-level *operational* way, by an idealised algorithm formulated in terms of implementation-level concepts (§2). This complicates reasoning about the behaviour of an application using a PSI database and, in particular, the correctness of a transaction chopping. To deal with this problem, we propose an alternative *axiomatic* specification of PSI that defines the consistency model declaratively by a set of axioms constraining client-visible events (§3). We prove that our axiomatic specification of PSI is equivalent to the existing operational one (§4). The axiomatic specification is instrumental in formulating and proving our transaction chopping criterion.

2 Operational Specification of PSI

We first present an operational specification of PSI, which is a simplification of the one originally proposed in [21]. It is given as an idealised algorithm that is formulated in terms of implementation-level concepts, such as replicas and messages, but nevertheless abstracts from many of the features that a realistic PSI implementation would have.

We consider a database storing ***objects*** $\mathsf{Obj} = \{x, y, \ldots\}$, which for simplicity we assume to be integer-valued. Clients interact with the database by issuing `read` and `write` operations on the objects, grouped into ***transactions***. We identify transactions by elements of $\mathsf{TId} = \{t_0, t_1, \ldots\}$. The database system consists of a set of ***replicas***, identified by $\mathsf{RId} = \{r_0, r_1, \ldots\}$, each maintaining a copy of all objects. Replicas may fail by crashing.

All client operations within a given transaction are initially executed at a single replica (though operations in *different* transactions can be executed at different replicas). When a client terminates the transaction, the replica decides whether to commit or abort it. To simplify the formal development, we assume that every transaction eventually terminates. If the replica decides to commit a transaction, it sends a message to all other replicas containing the ***transaction log***, which describes the updates done by the transaction. The replicas incorporate the updates into their state upon receiving the message. A transaction log has the form $(t, \texttt{start})\,(t, \texttt{write}(x_1, n_1)) \ldots (t, \texttt{write}(x_k, n_k))$, which gives the sequence of values $n_i \in \mathbb{N}$ written to objects $x_i \in \mathsf{Obj}$ by the transaction $t \in \mathsf{TId}$; the record $(t, \texttt{start})$ is added for convenience of future definitions. Transaction logs are ranged over by l, and we denote their set by Log.

We assume that every replica executes transactions locally without interleaving (this is a simplification in comparison to the original PSI specification [21] that makes the

<table>
<tr>
<td>operation <code>start</code>:
  requires $\mathsf{Current}[r] = \varepsilon$
  $t :=$ (unique identifier from TId)
  $\mathsf{Current}[r] := (t, \texttt{start})$</td>
<td colspan="2">operation <code>receive</code>(l):
  requires $\mathsf{Current}[r] = \varepsilon$
  requires $l = (t, \texttt{start}) \cdot \_$
  $\mathsf{Committed}[r] = \mathsf{Committed}[r] \cdot l$</td>
<td>operation <code>abort</code>:
  requires
  $\mathsf{Current}[r] =$
  $(t, \texttt{start}) \cdot \_$
  $\mathsf{Current}[r] := \varepsilon$</td>
</tr>
<tr>
<td>operation <code>write</code>(x, n):
  requires $\mathsf{Current}[r] = (t, \texttt{start}) \cdot \_$
  $\mathsf{Current}[r] := \mathsf{Current}[r] \cdot (t, \texttt{write}(x, n))$</td>
<td colspan="3" rowspan="2">operation <code>commit</code>:
  requires $\mathsf{Current}[r] = (t, \texttt{start}) \cdot \_$
  requires $\neg\exists x, r', t'.\ ((t, \texttt{write}(x, \_)) \in \mathsf{Current}[r]) \wedge$
  $(r \neq r') \wedge ((t', \texttt{write}(x, \_)) \in \mathsf{Committed}[r']) \wedge$
  $((t', \texttt{start}) \notin \mathsf{Committed}[r])$
  send $\mathsf{Current}[r]$ to all other replicas
  $\mathsf{Committed}[r] := \mathsf{Committed}[r] \cdot \mathsf{Current}[r]$
  $\mathsf{Current}[r] := \varepsilon$</td>
</tr>
<tr>
<td>operation <code>read</code>(x, n):
  requires $\mathsf{Current}[r] = (t, \texttt{start}) \cdot \_$
  requires $\texttt{write}(x, n)$ is the last write to x
  in $\mathsf{Committed}[r] \cdot \mathsf{Current}[r]$ or
  there is no such write and $n = 0$</td>
</tr>
</table>

Fig. 2. Pseudocode of the idealised PSI algorithm at replica r.

algorithm cleaner). This assumption allows us to maintain the state of a replica r in the algorithm by:

- $\mathsf{Current}[r] \in \mathsf{Log} \cup \{\varepsilon\}$—the log of the (single) transaction currently executing at r or an empty sequence ε, signifying that no transaction is currently executing; and
- $\mathsf{Committed}[r] \in \mathsf{Log}^*$—the sequence of logs of transactions that committed at r.

Initially $\mathsf{Current}[r] = \mathsf{Committed}[r] = \varepsilon$.

We give the pseudocode of the algorithm executing at a replica r in Figure 2. This describes the effects of ***operations*** executed at the replica, which come from the set

$$\mathsf{Op} = \{\texttt{start}, \texttt{receive}(l), \texttt{write}(x, n), \texttt{read}(x, n), \texttt{abort}, \texttt{commit} \mid l \in \mathsf{Log}, x \in \mathsf{Obj}, n \in \mathbb{N}\}$$

and are ranged over by o. The execution of the operations is atomic and is triggered by client requests or internal database events, such as messages arriving to the replica. The **requires** clauses give conditions under which an operation can be executed. For convenience of future definitions, operations do not return values. Instead, the value fetched by a `read` is recorded as its parameter; as we explain below, the **requires** clause for $\texttt{read}(x, n)$ ensures that the operation may only be executed when the value it fetches is indeed n. We use $\cdot$ for sequence concatenation, $\in$ to express that a given record belongs to a given sequence, and _ for irrelevant expressions.

When a client starts a transaction at the replica r (operation `start`), the database assigns it a unique identifier t and initialises $\mathsf{Current}[r]$ to signify that t is in progress. Since we assume that the replica processes transactions serially, in the idealised algorithm the transaction can start only if r is not already executing a transaction, as expressed by the **requires** clause. The operation $\texttt{receive}(l)$ executes when the replica receives a message l with the log of some transaction t, at which point it appends l to its log of committed transactions. A replica can receive a message only when it is not executing a transaction. When a client issues a write of n to an object x inside a transaction t, the corresponding record $(t, \texttt{write}(x, n))$ is appended to the log of the

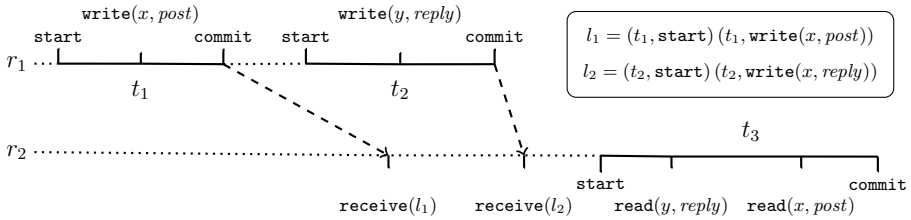

Fig. 3. An example execution of the operational PSI specification.

current transaction (operation $\texttt{write}(x, n)$). The **requires** clause ensures that a `write` operation can only be called inside a transaction. A client can read n as the value of an object x (operation $\texttt{read}(x, n)$) if it is the most recent value written to x at the replica according to the log of committed transactions concatenated with the log of the current one; if there is no such value, the client reads 0 (to simplify examples, in the following we sometimes assume different initial values).

If the current transaction aborts (operation `abort`), the $\mathsf{Current}[r]$ log is reset to be empty. Finally, if the current transaction commits (operation `commit`), its log is sent to all other replicas, as well as added to the log of committed transactions of the replica r. Crucially, as expressed by the second **requires** clause of `commit`, the database may commit a transaction t only if it passes the ***write-conflict detection*** check: there is no object x written by t that is also written by a concurrent transaction t', i.e., a transaction that has been committed at another replica r', but whose updates have not yet been received by r. If this check fails, the only option left for the database is to abort t using the operation `abort`.

In the algorithm we make certain assumptions about message delivery between replicas. First, every message is delivered to every replica at most once. Second, message delivery is ***causal***: if a replica sends a message l_2 after it sends or receives a message l_1, then every other replica will receive l_2 only after it receives or sends l_1; in this case we say that the transaction generating l_2 causally depends on the one generating l_1. This is illustrated by the execution of the algorithm depicted in Figure 3: due to causal delivery, the transaction t_3 that reads *reply* from y is also guaranteed to read *post* from x.

The operational specification of PSI is given by all sets of client-database interactions that can arise when executing the implementations of the operations in Figure 2 at each replica in the system. Due to the asynchronous propagation of updates between replicas, the specification of PSI allows non-serialisable behaviours, called ***anomalies***. We introduce structures to describe client-database interactions allowed by PSI and discuss its anomalies while presenting our declarative PSI specification, which is the subject of the next section.

3 Axiomatic Specification of PSI

Reasoning about PSI database behaviour using the operational specification may get unwieldy. It requires us to keep track of low-level information about the system state,

such as the logs at all replicas and the set of messages in transit. We then need to reason about how the system state is affected by a large number of possible interleavings of operations at different replicas. We now present a specification of PSI that is more declarative than the operational one and, in particular, does not refer to implementation-level details, such as message exchanges between replicas. It thus makes it easier to establish results about PSI, such as criteria for transaction chopping.

Our PSI specification is given by a set of *histories*, which describe all client-database interactions that this consistency model allows. To simplify presentation, our specification does not constrain the behaviour of aborted or ongoing transactions, so that histories only record operations inside committed transactions. Our specification also assumes that the database interface allows a client to group a finite number of transactions into a ***chain*** [24], which establishes an ordering on the transactions, similarly to a *session* [22]. Chains are needed for transaction chopping (§1) and can be implemented, e.g., by executing all transactions from a chain at the same replica.

To define histories and similar structures, we need to introduce some set-theoretic concepts. We assume a countably infinite set of ***events*** $\mathsf{Event} = \{e, f, g, \ldots\}$. A relation $R \subseteq E \times E$ on a set E is a ***strict partial order*** if it is transitive and irreflexive; it is an ***equivalence relation*** if it is reflexive, transitive and symmetric. For an equivalence relation $R \subseteq E \times E$ and $e \in E$, we let $[e]_R = \{f \mid (f, e) \in R\}$ be the ***equivalence class*** of e. A ***total order*** is a strict partial order such that for every two distinct elements e and f, the order relates e to f or f to e. We write $(e, f) \in R$ and $e \xrightarrow{R} f$ interchangeably.

Definition 1. *A **history** is a tuple $\mathcal{H} = (E, \mathsf{op}, \mathsf{co}, \sim)$, where:*

- $E \subseteq \mathsf{Event}$ *is a finite set of events, denoting reads and writes performed inside committed transactions.*
- $\mathsf{op} : E \to \{\mathtt{write}(x, n), \mathtt{read}(x, n) \mid x \in \mathsf{obj}, n \in \mathbb{N}\}$ *defines the operation each event denotes.*
- $\mathsf{co} \subseteq E \times E$ *is the **chain order**, arranging events in the same chain into the order in which a client submitted them to the database. We require that* co *be a union of total orders defined on disjoint subsets of E, which correspond to events in different chains.*
- $\sim \subseteq E \times E$ *is an equivalence relation grouping events in the same transaction. Since every transaction is performed by a single chain, we require that* co *totally order events within each transaction, i.e., those from $[e]_\sim$ for each $e \in E$. We also require that a transaction be contiguous in* co*:*

$$\forall e, f, g.\, e \xrightarrow{\mathsf{co}} f \xrightarrow{\mathsf{co}} g \wedge e \sim g \implies e \sim f \sim g.$$

Let Hist be the set of all histories. We denote components of a history $\mathcal{H}$ as in $E_\mathcal{H}$, and use the same notation for similar structures introduced in this paper. Our specification of PSI is given as a particular set of histories allowed by this consistency model. To define this set, we enrich histories with a *happens-before* relation, capturing causal relationships between events. In terms of the operational PSI specification, an event e happens before an event f if the information about e has been delivered to the replica performing f, and hence, can affect f's behaviour. The resulting notion of an *abstract execution* is similar to those used to specify weak shared-memory models [4].

$$\mathsf{op}(e) = \mathtt{read}(x, n) \implies \big(\exists f.\, \mathsf{op}(f) = \mathtt{write}(x, n) \wedge f \xrightarrow{\mathsf{hb}} e \wedge \neg\exists g.\, f \xrightarrow{\mathsf{hb}} g \xrightarrow{\mathsf{hb}} e \wedge \mathsf{op}(g) = \mathtt{write}(x, \_)\big) \vee \big(n = 0 \wedge \neg\exists f.\, f \xrightarrow{\mathsf{hb}} e \wedge \mathsf{op}(f) = \mathtt{write}(x, \_)\big) \quad \text{(Reads)}$$

$$\mathsf{co} \subseteq \mathsf{hb} \quad \text{(Chains)} \qquad \{(e', f') \mid e \xrightarrow{\mathsf{hb}} f \wedge e \not\sim f \wedge e' \sim e \wedge f \sim f'\} \subseteq \mathsf{hb} \quad \text{(Atomic)}$$

$$(e \neq f \wedge \{\mathsf{op}(e), \mathsf{op}(f)\} \subseteq \{\mathtt{write}(x, n) \mid n \in \mathbb{N}\}) \implies (e \xrightarrow{\mathsf{hb}} f \vee f \xrightarrow{\mathsf{hb}} e) \quad \text{(Wconflict)}$$

Fig. 4. Consistency axioms of PSI, stated for an execution $\mathcal{A} = ((E, \mathsf{op}, \mathsf{co}, \sim), \mathsf{hb})$. All free variables are universally quantified.

Definition 2. *An **abstract execution** is a pair $\mathcal{A} = (\mathcal{H}, \mathsf{hb})$ of a history $\mathcal{H}$ and the **happens-before** relation $\mathsf{hb} \subseteq E \times E$, which is a strict partial order.*

For example, Figure 5(a) shows an abstract execution, which corresponds to the execution of the operational specification in Figure 3 (as we formalise in §4). Our PSI specification is defined by *consistency axioms* (Figure 4), which constrain happens-before and other execution components and thereby describe the guarantees that a PSI database provides about transaction processing. We thus call this specification ***axiomatic***.

Definition 3. *An abstract execution $\mathcal{A}$ is **valid** if it satisfies the **consistency axioms** in Figure 4. We denote the set of all valid executions by AbsPSI and let the set of PSI histories be $\mathsf{HistPSI} = \{\mathcal{H} \in \mathsf{Hist} \mid \exists \mathsf{hb}.\, (\mathcal{H}, \mathsf{hb}) \in \mathsf{AbsPSI}\}$.*

The axiom (Reads) constrains the values fetched by a read using the happens-before relation: a read e from an object x has to return the value written by a hb-preceding write f on x that is most recent according to hb, i.e., not shadowed by another write g to x. If there is no hb-preceding write to x, then the read fetches the default value 0 (we sometimes use other values in examples). The axiom (Chains) establishes a causal dependency between events in the same chain (thus subsuming *session guarantees* [22]), and the transitivity of happens-before required in Definition 2 ensures that the database respects causality. For example, in the abstract execution in Figure 5(a), the chain order between the two writes induces an hb edge according to (Chains). Then, since hb is transitive, we must have an hb edge between the two operations on x and, hence, by (Reads), the read from x has to fetch *post*. There is no valid execution with a history where the read from y fetches *reply*, but the read from x fetches the default value. The operational specification ensures this because of causal message delivery.

The axiom (Atomic) ensures the ***atomic visibility*** of transactions: all writes by a transaction become visible to other transactions together. It requires that, if an event e happens before an event f in a different transaction, then all events e' in the transaction of e happen before all the events f' in the transaction of f. For example, (Atomic) disallows the execution in Figure 5(b), which is a variant of Figure 5(a) where the two writes are done in a single transaction and the order of the reads is reversed.

The axiom (Wconflict) states that the happens-before relation is total over write operations on a given object. Hence, the same object cannot be written by concurrent transactions, whose events are not related by happens-before. This disallows the ***lost update*** anomaly, illustrated by the execution in Figure 5(c). This execution could arise

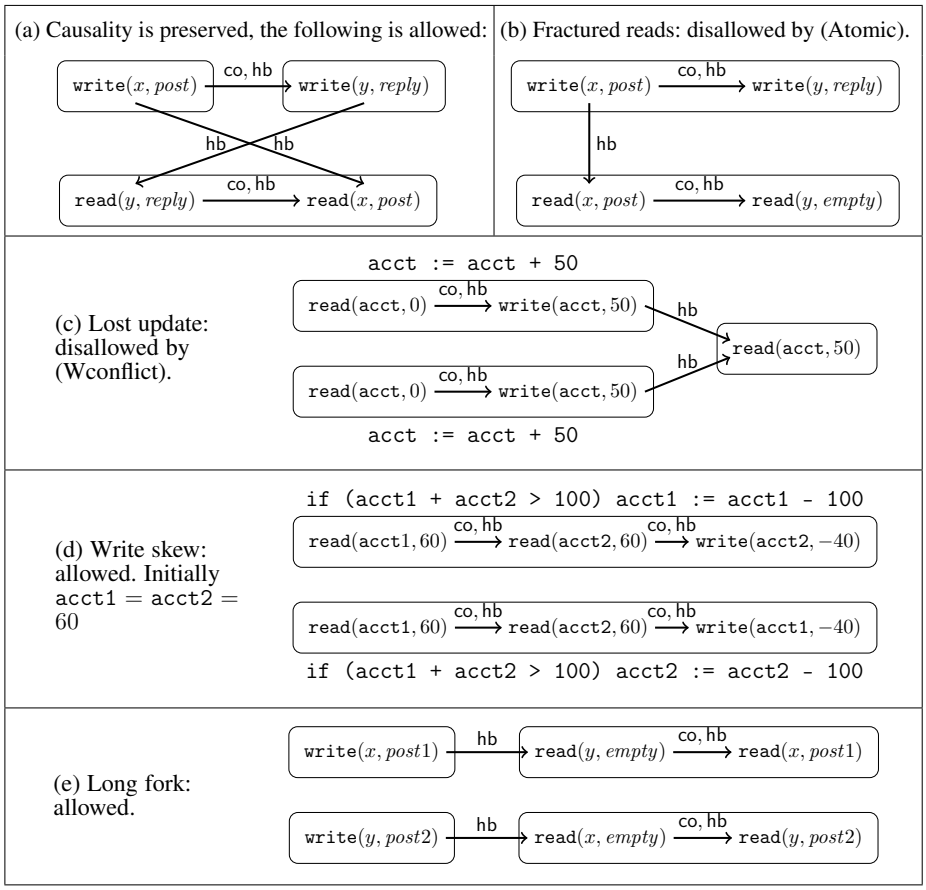

Fig. 5. Abstract executions illustrating PSI guarantees and anomalies. The boxes group events into transactions. We omit the transitive consequences of the co and hb edges shown.

from the code, also shown in the figure, that uses transactions to make deposits into an account; in this case, one deposit is lost. The execution violates (Wconflict): one of the transactions would have to hb-precede the other and, hence, read 50 instead of 0 from x. In the operational specification this anomaly is disallowed by the write-conflict detection, which would allow only one of the two concurrent transactions to commit.

Despite PSI disallowing many anomalies, it is weaker than serialisability. In particular, PSI allows the ***write skew*** anomaly, also allowed by the classical snapshot isolation [8]. We illustrate how our consistency axioms capture this by the valid execution in Figure 5(d), which could arise from the code also shown in the figure. Here each transaction checks that the combined balance of two accounts exceeds 100 and, if so, withdraws 100 from one of them. Both transactions pass the checks and make the withdrawals from different accounts, resulting in the combined balance going negative. The

operational specification allows this anomaly because the two transactions can be executed at different replicas and allowed to commit by the write-conflict detection check.

PSI also allows so-called ***long fork*** anomaly in Figure 5(e) [21], which we in fact already mentioned in §1. We have two concurrent transactions writing to x and y, respectively. A third transaction sees the write to x, but not y, and a fourth one sees the write to y, but not x. Thus, from the perspective of the latter two transactions, the two writes happen in different orders. It is easy to check that this outcome is not serialisable; in fact, it is also disallowed by the classical snapshot isolation. In the operational specification this anomaly can happen when each transaction executes at a separate replica, and the messages about the writes to x and y are delivered to the replicas executing the reading transactions in different orders.

4 Equivalence of the Specifications

We now show that the operational (§2) and axiomatic specifications (§3) are equivalent, i.e., the sets of histories they allow coincide. We start by introducing a notion of *concrete executions* of the operational PSI specification and using it to define the set of histories the specification allows. Concrete executions are similar to abstract ones of Definition 2, but describe *all* operations occurring at replicas as per Figure 2, including both client-visible and database-internal ones. We use the set-theoretic notions introduced before Definition 1.

Definition 4. *A **concrete execution** is a tuple $\mathcal{C} = (E, \mathsf{op}, \mathsf{repl}, \mathsf{trans}, \prec)$, where:*

- $E \subseteq \mathsf{Event}$ *is a finite set of events, denoting executions of operations in Figure 2.*
- $\mathsf{op} : E \to \mathsf{Op}$ *defines which of the operations in Figure 2 a given event denotes.*
- $\mathsf{repl} : E \to \mathsf{RId}$ *defines the replica on which the event occurs.*
- $\mathsf{trans} : E \to \mathsf{TId}$ *defines the transaction to which the event pertains.*
- $\prec \subseteq E \times E$ *is a total order, called **execution order**, in which events take place in the system.*

The set $\mathsf{ConcPSI}$ of concrete executions that can be produced by the algorithm in Figure 2 is defined as expected. Due to space constraints, we defer its formal definition to §C. Informally, the definition considers the execution of any sequence of operations in Figure 2 at arbitrary replicas, subject to the **requires** clauses and the constraints on message delivery mentioned in §2; the values of repl and trans are determined by the variables r and t in the code of operations in Figure 2. For example, Figure 3 can be viewed as a graphical depiction of a concrete execution from $\mathsf{ConcPSI}$, with the execution order given by the horizontal placement of events. For a $\mathcal{C} \in \mathsf{ConcPSI}$ and $e \in E_{\mathcal{C}}$, we write $e \triangleright_{\mathcal{C}} t : o \mathbin{@} r$ if $\mathsf{trans}_{\mathcal{C}}(e) = t$, $\mathsf{op}_{\mathcal{C}}(e) = o$ and $\mathsf{repl}_{\mathcal{C}}(e) = r$.

Definition 5. *The history of a concrete execution $\mathcal{C}$ is*

$$
\begin{aligned}
&\mathsf{history}(\mathcal{C}) = (E_{\mathcal{H}}, \mathsf{op}_{\mathcal{H}}, \mathsf{co}_{\mathcal{H}}, \sim_{\mathcal{H}}),\ \textit{where}\\
&\quad E_{\mathcal{H}} = \{e \in E_{\mathcal{C}} \mid \exists f \in E_{\mathcal{C}}, t \in \mathsf{TId}.\ (f \triangleright_{\mathcal{C}} t : \texttt{commit} \mathbin{@} \_) \wedge\\
&\qquad\qquad ((e \triangleright_{\mathcal{C}} t : \texttt{write}(\_,\_) \mathbin{@} \_) \vee (e \triangleright_{\mathcal{C}} t : \texttt{read}(\_,\_) \mathbin{@} \_))\};\\
&\quad \mathsf{op}_{\mathcal{H}} = (\textit{the restriction of } \mathsf{op}_{\mathcal{C}} \textit{ to } E_{\mathcal{H}});\\
&\quad \mathsf{co}_{\mathcal{H}} = \{(e, f) \in E_{\mathcal{H}} \times E_{\mathcal{H}} \mid \mathsf{repl}_{\mathcal{C}}(e) = \mathsf{repl}_{\mathcal{C}}(f) \wedge e \prec_{\mathcal{C}} f\}.\\
&\quad \sim_{\mathcal{H}} = \{(e, f) \in E_{\mathcal{H}} \times E_{\mathcal{H}} \mid \mathsf{trans}_{\mathcal{C}}(e) = \mathsf{trans}_{\mathcal{C}}(f)\};
\end{aligned}
$$

For example, the concrete execution in Figure 3 has the history shown in Figure 5(a). The history $\mathsf{history}(\mathcal{C})$ contains only the events describing reads and writes by the committed transactions in $\mathcal{C}$. To establish a correspondence between the operational and axiomatic specifications, we assume that chains are implemented by executing every one of them at a dedicated replica. Thus, we define the chain order $\mathsf{co}_{\mathcal{H}}$ as the order of events on each replica according to $\prec_{\mathcal{C}}$. This is, of course, an idealisation acceptable only in a specification. In a realistic implementation, multiple chains would be multiplexed over a single replica, or different transactions in a chain would be allowed to access different replicas [22]. We define the set of histories allowed by the operational PSI specification as $\mathsf{history}(\mathsf{ConcPSI})$, where we use the expected lifting of history to sets of executions. The following theorem (proved in §D) shows that this set coincides with the one defined by the axiomatic specification (Definition 3).

Theorem 1. $\mathsf{history}(\mathsf{ConcPSI}) = \mathsf{HistPSI}$.

5 Chopping PSI Transactions

In this section, we exploit the axiomatic specification of §3 to establish a criterion for checking the correctness of a ***chopping*** [20] of transactions executing on PSI. Namely, we assume that we are given a set of ***chain programs*** $\mathcal{P} = \{P_1, P_2, \ldots\}$, each defining the code of chains resulting from chopping the code of a single transaction. We leave the precise syntax of the programs unspecified, but assume that each P_i consists of k_i ***program pieces***, defining the code of the transactions in the chain. For example, for given `acct1`, `acct2` and `amnt`, Figure 1(b) defines a chain program resulting from chopping `transfer` in Figure 1(a). For a given `acct`, we can also create a chain program consisting of a single piece `lookup(acct)` in Figure 1(a). Let $\mathcal{P}^1$ consist of the programs for `lookup(acct1)`, `lookup(acct2)` and `transfer(acct1,acct2,amnt)`, and $\mathcal{P}^2$ of those for `transfer(acct1,acct2,amnt)` and `lookup2(acct1,acct2)`.

Following Shasha et al. [20], we make certain assumptions about the way clients execute chain programs. We assume that, if the transaction initiated by a program piece aborts, it will be resubmitted repeatedly until it commits, and, if a piece is aborted due to system failure, it will be restarted. We also assume that the client does not abort transactions explicitly.

In general, executing the chains $\mathcal{P}$ may produce more client-observable behaviours than if we executed every chain as a single PSI transaction. We propose a condition for checking that no new behaviours can be produced. To this end, we check that every valid abstract execution consisting of *fine-grained* transactions produced by the chains $\mathcal{P}$ can be *spliced* into another valid execution that has the same operations as the original one, but where all operations from each chain are executed inside a single *coarse-grained* transaction.

Definition 6. *Consider a valid abstract execution* $\mathcal{A} = ((E, \mathsf{op}, \mathsf{co}, \sim), \mathsf{hb}) \in \mathsf{AbsPSI}$ *and let* $\approx_{\mathcal{A}} = \mathsf{co} \cup \mathsf{co}^{-1} \cup \{(e, e) \mid e \in E\}$. *The execution* $\mathcal{A}$ *is* ***spliceable*** *if there exists* hb' *such that* $((E, \mathsf{op}, \mathsf{co}, \approx_{\mathcal{A}}), \mathsf{hb}') \in \mathsf{AbsPSI}$.

The definition groups fine-grained transactions in $\mathcal{A}$, identified by $\sim_{\mathcal{A}}$, into coarse-grained transactions, identified by $\approx_{\mathcal{A}}$, which consist of events in the same chain.

We now establish the core technical result of this section—a criterion for checking that an execution $\mathcal{A}$ is spliceable. From this *dynamic* criterion on executions we then obtain a *static* criterion for the correctness of chopping transaction code, by checking that all executions produced by the chain programs $\mathcal{P}$ are spliceable. We first need to define some auxiliary relations, derived from the happens-before relation in an abstract execution [2, 4].

Definition 7. *Given $\mathcal{A} \in \mathsf{AbsPSI}$, we define the **reads-from** $\mathsf{rf}_{\mathcal{A}}$, **version-order** $\mathsf{vo}_{\mathcal{A}}$ and **anti-dependency** $\mathsf{ad}_{\mathcal{A}}$ relations on $E_{\mathcal{A}}$ as follows:*

$$\begin{aligned}
e \xrightarrow{\mathsf{rf}_{\mathcal{A}}} f \iff & \exists x, n.\, e \xrightarrow{\mathsf{hb}_{\mathcal{A}}} f \wedge \mathsf{op}_{\mathcal{A}}(e) = \mathtt{write}(x, n) \wedge \mathsf{op}_{\mathcal{A}}(f) = \mathtt{read}(x, n) \wedge \\
& \neg\exists g.\, e \xrightarrow{\mathsf{hb}_{\mathcal{A}}} g \xrightarrow{\mathsf{hb}_{\mathcal{A}}} f \wedge \mathsf{op}_{\mathcal{A}}(g) = \mathtt{write}(x, _); \\
e \xrightarrow{\mathsf{vo}_{\mathcal{A}}} f \iff & \exists x.\, e \xrightarrow{\mathsf{hb}_{\mathcal{A}}} f \wedge \mathsf{op}_{\mathcal{A}}(e) = \mathtt{write}(x, _) \wedge \mathsf{op}_{\mathcal{A}}(f) = \mathtt{write}(x, _); \\
e \xrightarrow{\mathsf{ad}_{\mathcal{A}}} f \iff & \exists x.\, \mathsf{op}_{\mathcal{A}}(e) = \mathtt{read}(x, _) \wedge \mathsf{op}_{\mathcal{A}}(f) = \mathtt{write}(x, _) \wedge \\
& ((\exists g.\, g \xrightarrow{\mathsf{rf}_{\mathcal{A}}} e \wedge g \xrightarrow{\mathsf{vo}_{\mathcal{A}}} f) \vee (\neg\exists g.\, g \xrightarrow{\mathsf{rf}_{\mathcal{A}}} e)).
\end{aligned}$$

The reads-from relation determines the write e that a read f fetches its value from (uniquely, due to the axiom (Wconflict)). The version order totally orders all writes to a given object and corresponds to the order in which replicas find out about them in the operational specification. The anti-dependency relation [2] is more complicated. We have $e \xrightarrow{\mathsf{ad}_{\mathcal{A}}} f$ if the read e fetches a value that is overwritten by the write f according to $\mathsf{vo}_{\mathcal{A}}$ (the initial value of an object is overwritten by any write to this object).

Our criterion for checking that $\mathcal{A}$ is spliceable requires the absence of certain cycles in a graph with nodes given by the fine-grained transactions in $\mathcal{A}$ and edges generated using the above relations. The transactions are defined as equivalence classes $[e]_{\sim}$ of events $e \in E_{\mathcal{A}}$ (§3).

Definition 8. *Given $\mathcal{A} \in \mathsf{AbsPSI}$, its **dynamic chopping graph** $\mathsf{DCG}(\mathcal{A})$ is a directed graph whose set of nodes is $\{[e]_{\sim_{\mathcal{A}}} \mid e \in E_{\mathcal{A}}\}$, and we have an edge $([e]_{\sim_{\mathcal{A}}}, [f]_{\sim_{\mathcal{A}}})$ if and only if $[e]_{\sim_{\mathcal{A}}} \neq [f]_{\sim_{\mathcal{A}}}$ and one of the following holds: $e \xrightarrow{\mathsf{co}_{\mathcal{A}}} f$ (a **successor** edge); $f \xrightarrow{\mathsf{co}_{\mathcal{A}}} e$ (a **predecessor** edge); $e \xrightarrow{\mathsf{ad}_{\mathcal{A}} \setminus \approx_{\mathcal{A}}} f$ (an **anti-dependency** edge); or $e \xrightarrow{(\mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}}) \setminus \approx_{\mathcal{A}}} f$ (a **dependency** edge).*

*A **conflict** edge is one that is either a dependency or an anti-dependency. A directed cycle in the dynamic chopping graph is **critical** if it does not contain two occurrences of the same vertex, contains at most one anti-dependency edge, and contains a fragment of three consecutive edges of the form "conflict, predecessor, conflict".*

Theorem 2 (Dynamic Chopping Criterion). *An execution $\mathcal{A} \in \mathsf{AbsPSI}$ is spliceable if its dynamic chopping graph $\mathsf{DCG}(\mathcal{A})$ does not have critical cycles.*

We give a (non-trivial) proof of the theorem in §E. For example, the execution in Figure 6 satisfies the criterion in Theorem 2 and, indeed, we obtain a valid execution by grouping `withdraw` and `deposit` into a single transaction and adding the dotted happens-before edges.

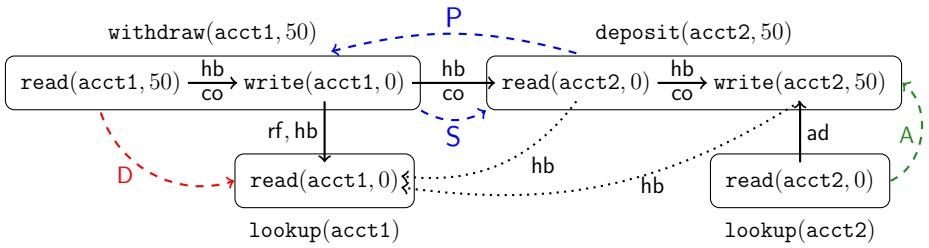

Fig. 6. An execution produced by the programs $\mathcal{P}^1$ and its derived relations. Initially $\texttt{acct1} = 50$ and $\texttt{acct2} = 0$. We omit the transitive consequences of the hb edges shown. The dashed edges show the dynamic chopping graph, with S, P, A, D denoting edge types. The dotted edges show additional happens-before edges that define a splicing of the execution (Definition 6).

We now use Theorem 2 to derive a static criterion for checking the correctness of code chopping given by $\mathcal{P}$. As is standard [12, 20], we formulate the criterion in terms of the sets of objects read or written by program pieces. Namely, for each chain program $P_i \in \mathcal{P}$ we assume a sequence

$$(R_1^i, W_1^i)\,(R_2^i, W_2^i)\,\ldots\,(R_{k_i}^i, W_{k_i}^i), \tag{1}$$

of ***read and write sets*** $R_j^i, W_j^i \subseteq \mathsf{Obj}$, i.e., the sets of all objects that can be, respectively, read and written by the j-th piece of P_i. For example, the `transfer(acct1,acct2,amnt)` chain in Figure 1(b) is associated with the sequence $(\{\texttt{acct1}\}, \{\texttt{acct1}\})\,(\{\texttt{acct2}\}, \{\texttt{acct2}\})$.

We consider a chopping defined by the programs $\mathcal{P}$ correct if all executions that they produce are spliceable. To formalise this, we first define when an execution can be produced by programs with read and write sets given by (1). Due to space constraints, we give the definition only informally.

Definition 9. *An abstract execution $\mathcal{A}$ **conforms** to a set of programs $\mathcal{P}$, if there is a one-to-one correspondence between every chain of transactions in $\mathcal{A}$ and a chain program $P_i \in \mathcal{P}$ whose read and write sets (1) cover the sets of objects read or written by the corresponding transactions in the chain.*

For example, the execution in Figures 6 conforms to the programs $\mathcal{P}^1$. Due to the assumptions about the way clients execute $\mathcal{P}$ that we made at the beginning of this section, the definition requires that every chain in an execution $\mathcal{A}$ conforming to $\mathcal{P}$ executes completely, and that all transactions in it commit. Also, for simplicity (and following [20]), we assume that every chain in $\mathcal{A}$ results from a distinct program in $\mathcal{P}$.

Definition 10. *Chain programs $\mathcal{P}$ are **chopped correctly** if every valid execution conforming to $\mathcal{P}$ is spliceable.*

We check the correctness of $\mathcal{P}$ by defining an analogue of the dynamic chopping graph from Definition 8 whose nodes are pieces of $\mathcal{P}$, rather than transactions in a given execution. Each piece is identified by a pair (i, j) of the number of a chain P_i and the piece's position in the chain.

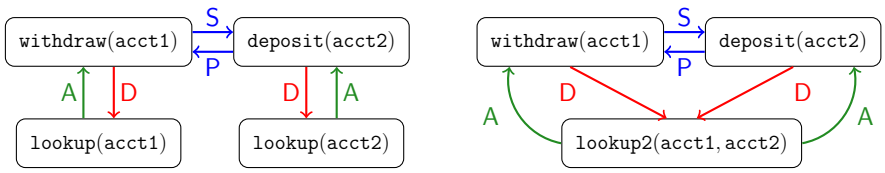

Fig. 7. Static chopping graphs for the programs (a) $\mathcal{P}^1$ and (b) $\mathcal{P}^2$.

Definition 11. *Given chain programs $\mathcal{P} = \{P_1, P_2, \ldots\}$ with read and write sets (1), the **static chopping graph** $\mathsf{SCG}(\mathcal{P})$ is a directed graph whose set of nodes is $\{(i, j) \mid i = 1..|\mathcal{P}|, j = 1..k_i\}$, and we have an edge $((i_1, j_1), (i_2, j_2))$ if and only if one of the following holds: $i_1 = i_2$ and $j_1 < j_2$ (a **successor** edge); $i_1 = i_2$ and $j_1 > j_2$ (a **predecessor** edge); $i_1 \neq i_2$, and $R^{i_1}_{j_1} \cap W^{i_2}_{j_2} \neq \emptyset$ (an **anti-dependency** edge); or $i_1 \neq i_2$, and $W^{i_1}_{j_1} \cap (R^{i_2}_{j_2} \cup W^{i_2}_{j_2}) \neq \emptyset$ (a **dependency** edge).*

For example, Figures 7(a) and 7(b) show the static chopping graph for the programs $\mathcal{P}^1$ and $\mathcal{P}^2$ respectively. There is a straightforward correspondence between $\mathsf{SCG}(\mathcal{P})$ and $\mathsf{DCG}(\mathcal{A})$ for an execution $\mathcal{A}$ conforming to $\mathcal{P}$: we have an (anti-)dependency edge between two pieces in $\mathsf{SCG}(\mathcal{P})$ if there *may* exist a corresponding edge in $\mathsf{DCG}(\mathcal{A})$ between two transactions resulting from executing the pieces, as determined by the read and write sets. Using this correspondence, from Theorem 2 we easily get a criterion for checking chopping correctness statically.

Corollary 1 (Static Chopping Criterion). *$\mathcal{P}$ is chopped correctly if $\mathsf{SCG}(\mathcal{P})$ does not contain any critical cycles.*

The graph in Figure 7(a) satisfies the condition of the corollary, whereas the one in Figure 7(b) does not. Hence, the corresponding chopping of `transfer` is correct, but becomes incorrect if we add `lookup2` (we provide an example execution illustrating the latter case in §A).

The criterion in Corollary 1 is more permissive than the one for chopping serialisable transactions previously proposed by Shasha et al. [20]. The latter does not distinguish between dependency and anti-dependency edges (representing them by a single type of a conflict edge) and between predecessor and successor edges (representing them by *sibling* edges). The criterion then requires the absence of any cycles containing both a conflict and a sibling edge. We illustrate the difference in Figure 8. The static chopping graph for the programs shown in the figure does not have critical cycles, but has a cycle with both a conflict and a sibling edge, and thus does not satisfy Shasha's criterion. We also show an execution produced by the programs: splicing the chains in it into single transactions (denoted by the dashed boxes) yields the execution in Figure 5(e) with a long fork anomaly. We provide a similar example for write skew (Figure 5(d)) in §A. Thus, the chopping criterion for PSI can be more permissive than the one for serialisability because of the anomalies allowed by the former consistency model.

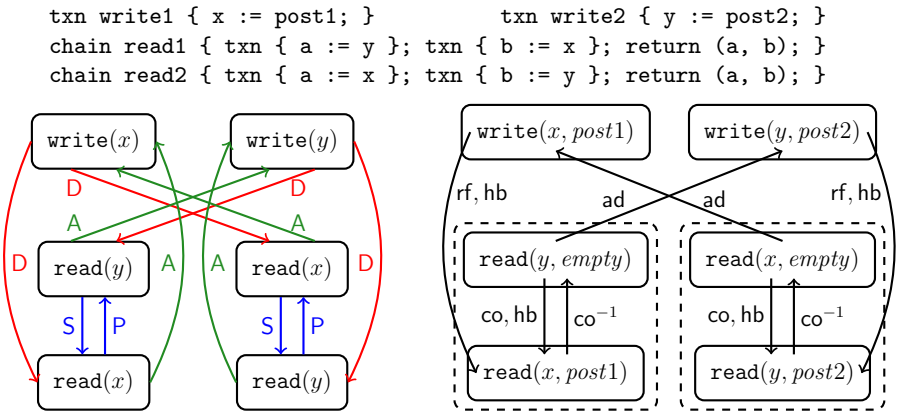

Fig. 8. An illustration of the difference between the chopping criteria for PSI and serialisability: programs, their static chopping graph and an example execution. The variables `a` and `b` are local.

Finally, we note that Theorem 2 and Corollary 1 do not make any assumptions about the structure of transactions, such as their commutativity properties, which may result in an excessive number of conflict edges in chopping graphs. These results can be strengthened to eliminate conflict edges between transactions whose effects commute, as done in [20, 24].

6 Related Work

Our criterion for the correctness of chopping PSI transactions was inspired by the criterion of Shasha et al. [20] for serialisable transactions. However, establishing a criterion for PSI is much more difficult than for serialisability. Due to the weakly consistent nature of PSI, reasoning about chopping correctness cannot be reduced to reasoning about a total serialisation order of events and requires considering intricate relationships between them, as Theorem 2 illustrates.

Our declarative specification of PSI uses a representation of executions more complex than the one in notions of strong consistency, such as serialisability [9] or linearizability [16]. This is motivated by the need to capture PSI anomalies. In proposing our specification, we built on the axiomatic approach to specifying consistency models, previously applied to eventual consistency [10] and weak shared-memory models [4]. In comparison to prior work, we handle a more sophisticated consistency model, including transactions with write-conflict detection. Our specification is also similar in spirit to the specifications of weak consistency models of relational databases of Adya's [2], which are based on the relations in Definition 7. While PSI could be specified in Adya's framework, we found that the specification based on the happens-before relation (Definition 2) results in simpler axioms and greatly eases proving the correspondence to the operational specification (Theorem 1) and the chopping criterion (Theorem 2).

Acknowledgements. We thank Hagit Attiya and Giovanni Bernardi for helpful discussions. This work was supported by EPSRC and an EU FET project ADVENT.

References

1. D. Abadi. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *IEEE Computer*, 45(2), 2012.
2. A. Adya. Weak consistency: A generalized theory and optimistic implementations for distributed transactions. PhD thesis, MIT, 1999.
3. Y. Afek, H. Avni, and N. Shavit. Towards consistency oblivious programming. In *OPODIS*, 2011.
4. J. Alglave. A formal hierarchy of weak memory models. *Formal Methods in System Design*, 41(2), 2012.
5. H. Attiya, A. Gotsman, S. Hans, and N. Rinetzky. A programming language perspective on transactional memory consistency. In *PODC*, 2013.
6. P. Bailis, A. Davidson, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Highly Available Transactions: virtues and limitations. In *VLDB*, 2014.
7. P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Scalable atomic visibility with RAMP transactions. In *SIGMOD*, 2014.
8. H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ANSI SQL isolation levels. In *SIGMOD*, 1995.
9. P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.
10. S. Burckhardt, A. Gotsman, H. Yang, and M. Zawirski. Replicated data types: specification, verification, optimality. In *POPL*, 2014.
11. S. Doherty, L. Groves, V. Luchangco, and M. Moir. Towards formally specifying and verifying transactional memory. *Formal Aspects of Computing*, 25(5), 2013.
12. A. Fekete, D. Liarokapis, E. O'Neil, P. O'Neil, and D. Shasha. Making snapshot isolation serializable. *ACM Trans. Database Syst.*, 30(2), 2005.
13. P. Felber, V. Gramoli, and R. Guerraoui. Elastic transactions. In *DISC*, 2009.
14. S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2), 2002.
15. R. Guerraoui and M. Kapalka. On the correctness of transactional memory. In *PPoPP*, 2008.
16. M. P. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3), 1990.
17. A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2), 2010.
18. S. Mu, Y. Cui, Y. Zhang, W. Lloyd, and J. Li. Extracting more concurrency from distributed transactions. In *OSDI*, 2014.
19. M. Saeida Ardekani, P. Sutra, and M. Shapiro. Non-monotonic snapshot isolation: Scalable and strong consistency for geo-replicated transactional systems. In *SRDS*, 2013.
20. D. Shasha, F. Llirbat, E. Simon, and P. Valduriez. Transaction chopping: Algorithms and performance studies. *ACM Trans. Database Syst.*, 20(3), 1995.
21. Y. Sovran, R. Power, M. K. Aguilera, and J. Li. Transactional storage for geo-replicated systems. In *SOSP*, 2011.
22. D. B. Terry, A. J. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. W. Welch. Session guarantees for weakly consistent replicated data. In *PDIS*, 1994.
23. L. Xiang and M. L. Scott. Software partitioning of hardware transactions. In *PPoPP*, 2015.
24. Y. Zhang, R. Power, S. Zhou, Y. Sovran, M. Aguilera, and J. Li. Transaction chains: Achieving serializability with low latency in geo-distributed storage systems. In *SOSP*, 2013.

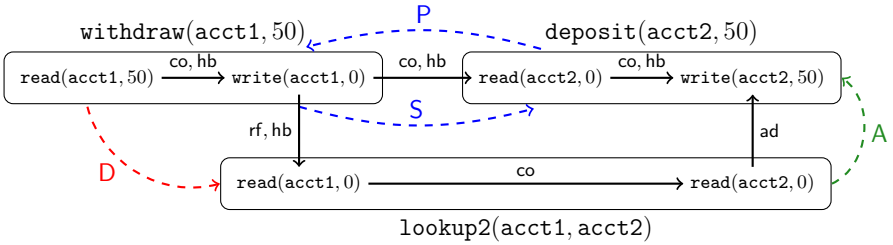

Fig. 9. An execution produced by the programs $\mathcal{P}^2$ and its derived relations. Initially $\texttt{acct1} = 50$ and $\texttt{acct2} = 0$. We omit the transitive consequences of the hb edges shown. The dashed edges show the dynamic chopping graph.

A Additional Examples

Figure 9 gives an example of an execution that does not satisfy the criterion Theorem 2: there is a critical cycle `lookup2`, `deposit`, `withdraw`, `lookup2`. In this case, the client notices the state with both accounts empty.

Figure 10 gives an example illustrating the difference between the chopping criteria for PSI and serialisability, analogous to the one in Figure 8. The static chopping graph shown again violates Shasha's criterion, but has no critical cycles. Splicing the execution in the figure, which conforms to the graph, yields an execution analogous to the write skew anomaly in Figure 5(d).

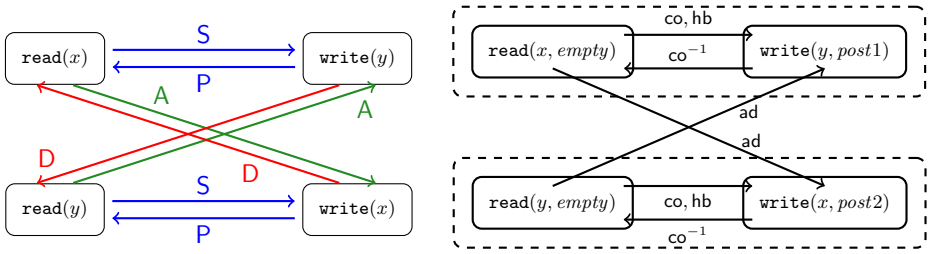

Fig. 10. An illustration of the difference between chopping criteria for PSI and serialisability related to the write skew anomaly (Figure 5(d)).

B Auxiliary Definitions and Lemmas

Definition 12 (Atomic Relations and Factoring). *Let E be a set of events, $\mathcal{R} \subseteq E \times E$ be a binary relation and $\sim \subseteq E \times E$ be an equivalence relation.*

The atomic closure *of $\mathcal{R}$, with respect to $\sim$, is defined to be $\sim;\mathcal{R};\sim$, and it is denoted as $\langle\mathcal{R}\rangle_\sim$. The relation $\mathcal{R}$ is said to be* atomic *if $\langle\mathcal{R}\rangle_\sim \subseteq \mathcal{R}$.*

The factoring *of $\mathcal{R}$, with respect to $\sim$, is defined to be $\sim;(\mathcal{R} \setminus \sim);\sim$, and it is denoted as $\langle\!\langle \mathcal{R} \rangle\!\rangle_\sim$.*

Lemma 1. *For any equivalence relation $\sim \subseteq E \times E$, the operator $\langle\cdot\rangle_\sim$ is a closure operator.*

Proof. We need to check that, for any relations $\mathcal{R}, \mathcal{S} \subseteq E \times E$,

$$\mathcal{R} \subseteq \langle \mathcal{R} \rangle_\sim \tag{Th1}$$

$$\mathcal{R} \subseteq \mathcal{S} \implies \langle \mathcal{R} \rangle_\sim \subseteq \langle \mathcal{S} \rangle_\sim \tag{Th2}$$

$$\langle \langle \mathcal{R} \rangle_\sim \rangle_\sim \subseteq \langle \mathcal{R} \rangle_\sim \tag{Th3}$$

First, suppose that $e \; \mathcal{R} \; f$. Since $\sim$ is reflexive, we obtain that $e \sim e \; \mathcal{R} \; f \sim f$, and by Definition 12 this means that $e \; \langle \mathcal{R} \rangle_\sim \; f$. We proved (Th1).

Next, suppose that $e \; \langle \mathcal{R} \rangle_\sim \; f$, and $\mathcal{R} \subseteq \mathcal{S}$. By Definition 12, there exist two events e', f' such that $e \sim e' \; \mathcal{R} \; f' \sim f$. Since $\mathcal{R} \subseteq \mathcal{S}$, we obtain that $e \sim e' \; \mathcal{S} \; f' \sim f$, hence $e \; \langle \mathcal{S} \rangle_\sim \; f$ by definition.

Finally, suppose that $e \; \langle \langle \mathcal{R} \rangle_\sim \rangle_\sim \; f$. By unfolding the definition of $\langle\cdot\rangle_\sim$ twice, we obtain that there exist e', e'', f', f'' such that $e \sim e' \sim e'' \; \mathcal{R} \; f'' \sim f' \sim f$. The transitivity of $\sim$ now gives that $e \sim e'' \; \mathcal{R} \; f'' \sim f$, which corresponds to $e \; \langle \mathcal{R} \rangle_\sim \; f$ by definition.

Lemma 2. *For any binary relation $\mathcal{R}$ and equivalence relation $\sim$, $\langle \mathcal{R} \backslash \sim \rangle_\sim \subseteq \mathcal{R}$ if and only if $\langle \mathcal{R} \backslash \sim \rangle_\sim \subseteq (\mathcal{R} \backslash \sim)$.*

Proof. Since $\mathcal{R} \backslash \sim \subseteq \mathcal{R}$, it is trivial to prove that if $\langle \mathcal{R} \backslash \sim \rangle_\sim \subseteq \mathcal{R} \backslash \sim$, then $\langle \mathcal{R} \backslash \sim \rangle_\sim \subseteq \mathcal{R} \backslash \sim \subseteq \mathcal{R}$.

Conversely, suppose that $\langle\!\langle \mathcal{R} \rangle\!\rangle_\sim \subseteq \mathcal{R}$. This means that $\langle \mathcal{R} \backslash \sim \rangle_\sim = \langle\!\langle \mathcal{R} \rangle\!\rangle_\sim \subseteq \mathcal{R}$. Thus, in order to prove that $\langle \mathcal{R} \backslash \sim \rangle_\sim \subseteq \mathcal{R} \backslash \sim$, it suffices to prove that whenever $e \; \langle \mathcal{R} \backslash \sim \rangle_\sim \; f$, then $e \not\sim f$. To see why this is true, note that $e \; \langle \mathcal{R} \backslash \sim \rangle_\sim \; f$ if and only if there exist e', f' such that $e \sim e'$, $f \sim f'$, $e' \not\sim f'$ and $e' \; \mathcal{R} \; f'$. Since $\sim$ is an equivalence, it follows that $e \not\sim f$, as we wanted to prove.

Definition 13. *Let $\mathcal{R}$ and $\sim$ be a relation and an equivalence relation, respectively, defined over a set of events E. The relation $\mathcal{R}_{/\sim} \subseteq E_{/\sim} \times E_{/\sim}$ (read $\mathcal{R}$ modulo $\sim$) is defined to be the least relation such that $[e]_\sim \; \mathcal{R}_{/\sim} \; [f]_\sim$ whenever $e \; \mathcal{R} \; f$.*

Lemma 3. *For any relation $\mathcal{R}$ and equivalence relation $\sim$ defined over a set of events E, $e \; \langle \mathcal{R} \rangle_\sim \; f$ if and only if $[e]_\sim \; \mathcal{R}_{/\sim} \; [f]_\sim$.*

Proof. We prove the two implications separately.

First note that, by definition, if $e \; \langle \mathcal{R} \rangle_\sim \; f$, then there exist e', f' such that $e \sim e' \; \mathcal{R} \; f' \sim f$. Using the definition of $E_{/\sim}$ and $\mathcal{R}_{/\sim}$, it follows that $[e]_\sim = [e']_\sim \; \mathcal{R}_{/\sim} \; [f']_\sim = [f]_\sim$, as we wanted to prove.

Next, suppose that $[e]_\sim \; \mathcal{R}_{/\sim} \; [f]_\sim$. Then there exist $e' \in [e]_\sim$ and $f' \in [f]_\sim$ such that $e' \; \mathcal{R} \; f'$. That is, $e \sim e' \; \mathcal{R} \; f' \sim f$, and by definition this means that $e \; \langle \mathcal{R} \rangle_\sim \; f$.

Corollary 2. *A relation $\mathcal{R}$ is atomic with respect to an equivalence $\sim$, if and only if $[e]_\sim \; \mathcal{R}_{/\sim} \; [f]_\sim \implies e \; \mathcal{R} \; f$.*

Proof. Because of Lemma 3, the RHS of the equivalence in the lemma means that for all e, f,

$$e \langle \mathcal{R} \rangle_\sim f \implies e \mathcal{R} f.$$

This is precisely the definition of $\mathcal{R}$ being atomic.

Lemma 4. *If $\mathcal{R}, \mathcal{S}$ are two atomic relations, with respect to some equivalence $\sim$, then $\mathcal{R}; \mathcal{S}$ is atomic with respect to $\sim$.*

Proof. It suffices to prove that whenever $[e]_\sim \ (\mathcal{R}; \mathcal{S})_{/\sim} \ [f]_\sim$, then $e \ (\mathcal{R}; \mathcal{S}) \ f$. The result follows from Corollary 2. Suppose that $[e]_\sim \ (\mathcal{R}; \mathcal{S})_{/\sim} \ [f]_\sim$; by definition, there exist e', f, g such that $e \sim e' \ \mathcal{R} \ g \ \mathcal{S} \ f' \sim f$. By using the atomicity of $\mathcal{R}$, and the fact that $g \sim g$, we obtain that $e \ \mathcal{R} \ g$; similarly, we can prove that $g \ \mathcal{S} \ f$ by employing the atomicity of $\mathcal{S}$. Therefore, $e \ (\mathcal{R}; \mathcal{S}) \ f$.

Corollary 3. *For any atomic relation $\mathcal{R}$, with respect to an equivalence $\sim$, $\mathcal{R}^+$ is atomic with respect to $\sim$.*

Proof. First note that, if $\mathcal{R}$ is atomic with respect to $\sim$, then $\mathcal{R}^k$ is atomic with respect to $\sim$ for any $k > 0$. This can be proved by performing a simple induction over k, using Lemma 4 in the inductive step.

Now suppose that $e \sim e' \ \mathcal{R}^+ \ f' \sim f$ for some e, e', f, f'. This means that there exists an index $k > 0$ such that $e \sim e' \ \mathcal{R}^k \ f' \sim f$, hence $e \ \mathcal{R}^k \ f$ by the atomicity of $\mathcal{R}^k$. It follows that $e \ \mathcal{R}^+ \ f$.

Corollary 4. *For any atomic relation $\mathcal{R}$, with respect to an equivalence $\sim$, we have that $\langle \mathcal{R}^+ \rangle_\sim = \langle \mathcal{R} \rangle_\sim^+ = \mathcal{R}^+$.*

Also, ${\mathcal{R}^+}_{/\sim} = (\mathcal{R}_{/\sim})^+$.

Proof. Since $\mathcal{R}$ is atomic, we know that $\mathcal{R} = \langle \mathcal{R} \rangle_\sim$, hence $\mathcal{R}^+ = \langle \mathcal{R} \rangle_\sim^+$. Also, by Corollary 3 we know that $\mathcal{R}^+$ is atomic, hence $\langle \mathcal{R}^+ \rangle_\sim = \mathcal{R}^+$.

To prove that ${\mathcal{R}^+}_{/\sim} = (\mathcal{R}_{/\sim})^+$, it suffices to show that ${\mathcal{R}^k}_{/\sim} = (\mathcal{R}_{/\sim})^k$ for any $k \geq 1$. This can be proved by induction on k, using Corollary 2.

Definition 14. *Let $\mathcal{R} \subseteq E \times E$ be a relation over a set of events E. A* cycle *for $\mathcal{R}$ is a sequence of events $e_0 \cdots e_n$ with $n \geq 1$ such that $e_i \ \mathcal{R} \ e_{i+1}$ for any $i = 0, \cdots, n-1$ and $e_0 = e_n$.*

A cycle $e_0 \cdots e_n$ is said to be simple if for any i, j such that $0 \leq i, j < n$, $e_i = e_j$ implies that $i = j$.

A relation $\mathcal{R}$ is cyclic *if there exists a cycle for it,* acyclic *otherwise.*

Lemma 5. *A relation $\mathcal{R}$ is acyclic if and only if $\mathcal{R}^+$ is irreflexive.*

Proof. We prove the contrapositive statement: $\mathcal{R}$ has a cycle if and only if there exists an element e such that $e \ \mathcal{R}^+ \ e$.

First, suppose that $\mathcal{R}$ has a cycle $e_0 \cdots e_n$. By definition, this means that $e_0 \ \mathcal{R}^n \ e_n$, and also that $e_0 = e_n$. That is, $e_0 \ \mathcal{R}^+ \ e_0$.

Next, suppose that $e \ \mathcal{R}^+ \ e$ for some $e \in E$. That is, there exists $n > 0$ such that $e \ \mathcal{R}^n \ e$. By definition, $e \ \mathcal{R}^n \ e$ means that there exist $e_1, \cdots, e_{n-1}$ such that $e_0 \ \mathcal{R} \ e_1 \ \mathcal{R} \cdots \mathcal{R} \ e_{n-1} \ \mathcal{R} \ e$. We have constructed a cycle for $\mathcal{R}$, namely $ee_1 \cdots e_{n-1}e$.

Lemma 6. *Let $\mathcal{R}$ be an atomic relation with respect to an equivalence relation $\sim$. Then $\mathcal{R}$ is acyclic if and only if both $\mathcal{R} \cap \sim$ and $(\mathcal{R}_{/\sim} \setminus \mathsf{Id})$ are acyclic.*

Proof. We prove the contrapositive statement. First, suppose that $\mathcal{R}$ has a cycle $e_0 \cdots e_n$. By definition, for any $i = 0, \cdots, n-1$, $e_i \ \mathcal{R} \ e_{i+1}$, and $e_0 = e_n$. Since $\mathcal{R} = (\mathcal{R} \cap \sim) \cup (\mathcal{R} \setminus \sim)$, we have that for any $i = 0, \cdots, n$ either $e_i \ (\mathcal{R} \cap \sim) \ e_{i+1}$ or $e_i \ (\mathcal{R} \setminus \sim) \ e_{i+1}$. Note that if there exist no index i such that $e_i \ (\mathcal{R} \setminus \sim) \ e_{i+1}$, then it has to be the case that

$$e_0 \ (\mathcal{R} \cap \sim) \ e_1 \ (\mathcal{R} \cap \sim) \cdots (\mathcal{R} \cap \sim) \ e_{n-1} \ (\mathcal{R} \cap \sim) \ e_n$$

and, since $e_0 = e_n$, we have shown that the relation $(\mathcal{R} \cap \sim)$ is cyclic. Therefore, we can safely assume that there exists at least an index i such that $e_i \ (\mathcal{R} \setminus \sim) \ e_{i+1}$. Let $\{e_{j_0}, \cdots, e_{j_k} \mid \forall h = 0, \cdots, k.\ 0 \leq j_h \leq n \wedge \forall h' = 0, \cdots, k. h < h' \implies j_h < j_{h'}\}$ be the largest set of events such that, for any $h = 0, \cdots, k-1$, $e_{j_h} \ (\mathcal{R} \setminus \sim) \ e_{j_{h+1}}$; notice that such a set is non-empty by assumption. We also have that

$$[e_0]_\sim = [e_{j_0}]_\sim \ (\mathcal{R} \setminus \sim)_{/\sim} \ [e_{j_1}]_\sim \ (\mathcal{R} \setminus \sim)_{/\sim} \cdots (\mathcal{R} \setminus \sim)_{/\sim} \ [e_{j_k}]_\sim = [e_n]_\sim$$

Since $e_0 = e_n$ by hypothesis, it is straightforward to infer from the above that $[e_{j_0}]_\sim = [e_{j_k}]_\sim$. Also, a consequence of $[e_{j_h}]_\sim \ (\mathcal{R} \setminus \sim)_{/\sim} \ [e_{j_{h+1}}]_\sim$ is that $[e_{j_h}]_\sim \ \mathcal{R}_{/\sim} \ [e_{j_{h+1}}]_\sim$, and $[e_{j_h}]_\sim \neq [e_{j_{h+1}}]_\sim$. We have proved that

$$[e_{j_0}]_\sim \ (\mathcal{R}_{/\sim} \setminus \mathsf{Id}) \ [e_{j_1}]_\sim \ (\mathcal{R}_{/\sim} \setminus \mathsf{Id}) \cdots (\mathcal{R}_{/\sim} \setminus \mathsf{Id}) \ [e_{j_k}]_\sim$$

and $[e_{j_0}]_\sim = [e_{j_k}]_\sim$. Thus, we have found a cycle for $(\mathcal{R}_{/\sim} \setminus \mathsf{Id})$.

Next, suppose that the relation $\mathcal{R} \cap \sim$ has a cycle. It is trivial to see that, in this case, $\mathcal{R}$ also has a cycle. It remains to prove that, if $(\mathcal{R}_{/\sim} \setminus \mathsf{Id})$ has a cycle, then $\mathcal{R}$ also has a cycle. To this end, suppose that $(\mathcal{R}_{/\sim} \setminus \mathsf{Id})$ has a cycle $e_0 \cdots e_n$; by definition, we have that

$$[e_0]_\sim \ (\mathcal{R}_{/\sim} \setminus \mathsf{Id}) \ [e_1]_\sim \ (\mathcal{R}_{/\sim} \setminus \mathsf{Id}) \cdots (\mathcal{R}_{/\sim} \setminus \mathsf{Id}) \ [e_n]_\sim$$

from which we can infer that

$$[e_0]_\sim \ (\mathcal{R}_{/\sim}) \ [e_1]_\sim \ (\mathcal{R}_{/\sim}) \cdots (\mathcal{R}_{/\sim}) \ [e_n]_\sim$$

Since $\mathcal{R}$ is atomic by hypothesis, we can employ Corollary 2 to obtain that

$$e_0 \ \mathcal{R} \ e_1 \ \mathcal{R} \cdots \mathcal{R} \ e_n$$

which constitutes a cycle for $\mathcal{R}$.

Lemma 7. *Let $\sim$ be an equivalence over a set of events E, and $\mathcal{S}$ be a relation such that $\mathcal{S} \subseteq \sim$. Then $\mathcal{S}; \langle \mathcal{R} \backslash \sim \rangle_\sim; \mathcal{S} \subseteq \langle \mathcal{R} \backslash \sim \rangle_\sim$.*

Proof. Since, $\mathcal{S} \subseteq \sim$ by hypothesis, we have that

$$\mathcal{S}; \langle \mathcal{R} \backslash \sim \rangle_\sim; \mathcal{S} \subseteq \sim; \langle \mathcal{R} \backslash \sim \rangle_\sim; \sim = \langle \langle \mathcal{R} \backslash \sim \rangle_\sim \rangle_\sim = \langle \mathcal{R} \backslash \sim \rangle_\sim$$

where we applied Lemma 1 when proving the equality $\langle \langle \mathcal{R} \backslash \sim \rangle_\sim \rangle_\sim = \langle \mathcal{R} \backslash \sim \rangle_\sim$.

Corollary 5. *Let $\mathcal{S} \subseteq \sim$ be a transitive relation, where $\sim$ is an equivalence over a set of events E. Then, for any relation $\mathcal{R}$, we have that $(\mathcal{S} \cup \langle \mathcal{R} \backslash \sim \rangle_\sim)^+ = \mathcal{S} \cup \langle \mathcal{R} \backslash \sim \rangle_\sim^+$.*

Proof. Recall that, for any two relations $\mathcal{R}, \mathcal{S}$, we have that $\mathcal{S} \subseteq \mathcal{S}^+$ and $(\mathcal{R}^+ \cup \mathcal{S}^+) \subseteq (\mathcal{R} \cup \mathcal{S})^+$. Therefore, it is straightforward to show that

$$\mathcal{S} \cup \langle \mathcal{R} \backslash \sim \rangle_\sim^+ \subseteq (\mathcal{S}^+ \cup \langle \mathcal{R} \backslash \sim \rangle_\sim^+) \subseteq (\mathcal{S} \cup \langle \mathcal{R} \backslash \sim \rangle_\sim)^+$$

It remains to prove that $(\mathcal{S} \cup \langle \mathcal{R} \backslash \sim \rangle_\sim)^+ \subseteq \mathcal{S} \cup \langle \mathcal{R} \backslash \sim \rangle_\sim^+$. Suppose that $e \ (\mathcal{S} \cup \langle \mathcal{R} \backslash \sim \rangle_\sim)^+ \ f$. This means that there exist a sequence of events $e_0, e_1, \cdots, e_n$ such that for any $e_0 = e$, $e_n = f$ and for any $i = 0, \cdots, n-1$, $e_i \ (\mathcal{S} \cup \langle \mathcal{R} \backslash \sim \rangle_\sim) \ e_{i+1}$. Note that if it were $e_i \ \mathcal{S} \ e_{i+1}$ for any $i = 0, \cdots, n-1$, then we would have that $e = e_0 \ \mathcal{S}^+ \ e_n = f$, hence $e \ \mathcal{S} \ f$ by the transitivity of $\mathcal{S}$. Otherwise, there exists at least one index i such that $e_i \ \langle \mathcal{R} \backslash \sim \rangle_\sim \ e_{i+1}$. In this case we have that $e \ (\mathcal{S}^*; \langle \mathcal{R} \backslash \sim \rangle_\sim; \mathcal{S}^*)^k \ f$ for some k such that $0 < k \leq n$; that is, $e \ (\mathcal{S}^*; \langle \mathcal{R} \backslash \sim \rangle_\sim; \mathcal{S}^*)^+ \ f$. We can now apply Lemma 7 to obtain that $e \ \langle \mathcal{R} \backslash \sim \rangle_\sim^+ \ f$, which concludes the proof.

Lemma 8. *For any relation $\mathcal{R}$ and equivalence $\sim$, over a set of events E, $\langle \mathcal{R} \backslash \sim \rangle_\sim = \langle \mathcal{R} \rangle_\sim \backslash \sim$.*

Proof. By Lemma 3, we have that $e \ \langle (\mathcal{R} \backslash \sim) \rangle_\sim \ f$ if and only if $[e]_\sim \ \mathcal{R} \backslash \sim_{/\sim} \ [f]_\sim$, which is true if and only if $[e]_\sim \ \mathcal{R}_{/\sim} \ [f]_\sim$ and $[e]_\sim \neq [f]_\sim$. Again, $[e]_\sim \ \mathcal{R}_{/\sim} \ [f]_\sim$ if and only if $e \ \langle \mathcal{R} \rangle_\sim \ f$ by Lemma 3. Thus we have proved that $e \ \langle \mathcal{R} \backslash \sim \rangle_\sim \ f$ if and only if $e \ \langle \mathcal{R} \rangle_\sim \ f$ and $e \not\sim f$, or equivalently that $\langle \mathcal{R} \backslash \sim \rangle_\sim = \langle \mathcal{R} \rangle_\sim \backslash \sim$.

Lemma 9. *Let E be a set of events, and let $\sim, \approx$ be two equivalences over E such that $\sim \subseteq \approx$. Then*

1. $\langle \approx \rangle_\sim \subseteq \approx$,
2. $[e]_\sim \ (\approx_{/\sim}) \ [f]_\sim$ *if and only if* $e \approx f$,
3. $\approx_{/\sim} \subseteq E_{/\sim} \times E_{/\sim}$ *is an equivalence.*

Proof.

1. Note that $\sim \subseteq \approx$ by hypothesis, so that $\langle \approx \rangle_\sim = \sim; \approx; \sim \subseteq \approx; \approx; \approx = \approx$.
2. If $e \approx f$, then $[e]_\sim \ (\approx_{/\sim}) \ [f]_\sim$ by definition. Conversely, if $[e]_\sim \ (\approx_{/\sim}) \ [f]_\sim$ then there exist $e' \sim e$, $f' \sim f$ such that $e' \approx f'$. Since $\sim \subseteq \approx$, and $\approx$ is an equivalence, it follows that $e \approx f$.
3. This is a straightforward consequence of (2) above, and the fact that $\approx$ is an equivalence.

Lemma 10. *Let $\mathcal{R}, \sim, \approx$ be a relation and two equivalences, respectively, over a set of events E. Suppose that $\sim \subseteq \approx$. Then, for any $e, f \in E$, $e \ \langle \mathcal{R} \backslash \approx \rangle_\approx \ f \iff [e]_\sim \ \langle \mathcal{R}_{/\sim} \backslash (\approx_{/\sim}) \rangle_{\approx_{/\sim}} \ [f]_\sim$.*

Proof. Suppose that $e \ \langle \mathcal{R} \backslash \approx \rangle_\approx \ f$ for some $e, f \in E$. Then there exist $e' \approx e, f' \approx f$ such that $e' \mathcal{R} f'$ and $e' \not\approx f'$. By Lemma 9 (2), we know that $e' \not\approx f'$ implies that $\neg \left([e']_\sim \ (\approx_{/\sim}) \ [f']_\sim \right)$. Also, the same Lemma ensures that $\left([e]_\sim \ (\approx_{/\sim}) \ [e']_\sim \right)$, and

$\big([f]_\sim\ (\approx_{/\sim})\ [f']_\sim\big)$. Finally, the definition of $\mathcal{R}_{/\sim}$ gives that $[e']_\sim\ (\mathcal{R}_{/\sim})\ [f']_\sim$. Therefore, we have proved that $[e]_\sim\ (\approx_{/\sim})\ [e']_\sim\ (\mathcal{R}_{/\sim})\backslash(\approx_{/\sim})\ [f']_\sim\ (\approx_{/\sim})\ [f]_\sim$, that is $[e]_\sim\ \langle\mathcal{R}_{/\sim}\backslash(\approx_{/\sim})\rangle_{\approx_{/\sim}}\ [f]_\sim$.

Next, suppose that $[e]_\sim\ \langle\mathcal{R}_{/\sim}\backslash(\approx_{/\sim})\rangle_{\approx_{/\sim}}\ [f]_\sim$ for some $e, f \in E$. By definition, there exist $[e']_\sim\ (\approx_{/\sim})\ [e]_\sim$, $[f']_\sim\ (\approx_{/\sim})\ [f]_\sim$, such that $\neg\big([e']_\sim\ (\approx_{/\sim})[f']_\sim\big)$ and $[e']_\sim\ \mathcal{R}_{/\sim}\ [f']_\sim$. By Lemma 9 this implies that $e' \approx e, f' \approx f$, and $e' \not\approx f'$. By definition of $[e']_\sim\ \mathcal{R}_{/\sim}\ [f']_\sim$, there exist $e'' \sim e', f'' \sim f'$ such that $e''\ \mathcal{R}\ f''$; since $\sim\subseteq\approx$ by hypothesis, this also implies that $e' \approx e''$, $f'' \approx f'$, and since $e' \not\approx f'$ we also have that $e'' \not\approx f''$. We have proved that $e \approx e''$, $e''\ \mathcal{R}\ f''$, $f'' \approx f$ and $e'' \not\approx f''$. That is, $e\ \langle\mathcal{R}\backslash\approx\rangle_\approx\ f$.

Definition 15 (Previous Writes). *Let $\mathcal{A}$ be an abstract execution, and for any event $f \in E_\mathcal{A}$ and object $x \in \mathsf{Obj}$ the set of the* previous writes *of f over x, denoted as $\mathsf{PW}^x_\mathcal{A}(f)$, is defined as $\{e \in E_\mathcal{A} \mid e \xrightarrow{\mathsf{hb}_\mathcal{A}} f \wedge \mathsf{op}(e) = \mathtt{write}(x, \_)\}$.*

Lemma 11. *Let $\mathcal{A}$ be any abstract execution that satisfies* (Wconflict)*. Then*

$$\forall e, x.(\mathsf{PW}^x_\mathcal{A}(e) = \emptyset \iff \max_{\mathsf{hb}_\mathcal{A}}(\mathsf{PW}^x_\mathcal{A}(e))\uparrow) \tag{WWMax}$$

where $\uparrow$ denotes undefinedness.

Proof. Recall that the maximum of an empty set is not defined, so that if $\mathsf{PW}^x_\mathcal{A}(e) = \emptyset$ then $\max_{\mathsf{hb}_\mathcal{A}}(\mathsf{PW}^x_\mathcal{A}(e))\uparrow$. Conversely, suppose that $\max_{\mathsf{hb}_\mathcal{A}}(\mathsf{PW}^x_\mathcal{A}(e))\uparrow$. Since E is finite by hypothesis, then so is the set $\mathsf{PW}^x_\mathcal{A}(e)$ is finite. Also, by (Wconflict) this set is also totally ordered with respect to $\mathsf{hb}_\mathcal{A}$. However, a finite total order does not admit a maximum element only when it is defined on an empty set, that is $\mathsf{PW}^x_\mathcal{A}(e) = \emptyset$.

Lemma 12 (Alternative Characterisation of (Reads)**).** *Let $\mathcal{A}$ be an abstract execution that satisfies* (Wconflict)*. Then $\mathcal{A}$ satisfies* (Reads) *if and only if it satisfies*

$$\forall e, x, n.\mathsf{op}(e) = \mathtt{read}(x, n) \implies$$
$$\big((\mathsf{PW}^x_\mathcal{A}(e)) = \emptyset \implies n = 0) \wedge (\mathsf{op}(\max_{\mathsf{hb}_\mathcal{A}}(\mathsf{PW}^x_\mathcal{A})) = \mathtt{write}(x, m) \implies m = n)\big) \tag{RFmax}$$

Proof. First suppose that $\mathcal{A}$ satisfies both (Wconflict) and (Reads). Let e be an event such that $\mathsf{op}(e) = \mathtt{read}(x, n)$ for some object x and value n. By (Reads) there are two possible cases:

- **(i)** $f \xrightarrow{\mathsf{hb}_\mathcal{A}} e$ for some f such that $\mathsf{op}(f) = \mathtt{write}(x, n)$ and **(ii)** $\neg\exists g.f \xrightarrow{\mathsf{hb}_\mathcal{A}} g \xrightarrow{\mathsf{hb}_\mathcal{A}} e \wedge \mathsf{op}(g) \neq \mathsf{op}(g) = \mathtt{write}(x, \_)$. Condition **(i)** states that $f \in \mathsf{PW}^x_\mathcal{A}(e)$, which together with Condition **(ii)** gives $f = \max_{\mathsf{hb}_\mathcal{A}}(\mathsf{PW}^x_\mathcal{A}(e))$. Therefore, $\mathsf{op}(\max_{\mathsf{hb}_\mathcal{A}}(\mathsf{PW}^x_\mathcal{A}(e)) = \mathsf{op}(f) = \mathtt{write}(x, n)$. We have proved that $\mathsf{op}(\max_{\mathsf{hb}_\mathcal{A}}(\mathsf{PW}^x_\mathcal{A})) = \mathtt{write}(x, m) \implies m = n$. Also, by Lemma 11, we know that $\mathsf{PW}^x_\mathcal{A}(e) \neq \emptyset$, hence the implication $(\mathsf{PW}^x_\mathcal{A}(e)) = \emptyset \implies n = 0$ follows trivially. Now it is easy to see that $\mathcal{A}$ satisfies (RFmax).

– $n = 0 \wedge \neg\exists f.f \xrightarrow{\mathsf{hb}_\mathcal{A}} e$; in this case we have that $\mathsf{PW}^x_\mathcal{A}(e) = \emptyset$, so that $(\mathsf{PW}^x_\mathcal{A}(e)) = \emptyset \implies n = 0$). By Lemma 11 we know that $\mathsf{op}(\max_{\mathsf{hb}_\mathcal{A}}(\mathsf{PW}^x_\mathcal{A}))\uparrow$, hence $\mathsf{op}(\max_{\mathsf{hb}_\mathcal{A}}(\mathsf{PW}^x_\mathcal{A})) \neq \mathtt{write}(x,\_)$. Therefore, the implication $\mathsf{op}(\max_{\mathsf{hb}_\mathcal{A}}(\mathsf{PW}^x_\mathcal{A})) = \mathtt{write}(x,m) \implies m = n$ is trivially satisfied, and again we have proved that (RFmax) is satisfied by $\mathcal{A}$.

Conversely, suppose that $\mathcal{A}$ satisfies both (Wconflict) and (RFmax). Let $e \in E_\mathcal{A}$ be an event such that $\mathsf{op}(e) = \mathtt{read}(x,n)$. According to Lemma 11 there are two possible cases:

– $\mathsf{PW}^x_\mathcal{A}(e) = \emptyset$, in which case $n = 0$ by (RFmax). By definition of $\mathsf{PW}^x_\mathcal{A}$, we know that $\neg\exists f.\mathsf{op}(f) = \mathtt{write}(x,\_) \wedge f \xrightarrow{\mathsf{hb}_\mathcal{A}} e$, from which (Reads) follows immediately,
– $\max_{\mathsf{hb}_\mathcal{A}}(\mathsf{PW}^x_\mathcal{A}(e)) = f$ for some event f. By definition, $\mathsf{op}(f) = \mathtt{write}(x,m)$ for some m; also, by (RFmax), $m = n$. We also have that $f \in \mathsf{PW}^x_\mathcal{A}(e)$, that is $f \xrightarrow{\mathsf{hb}_\mathcal{A}} e$. Also, whenever g is an event such that $g \neq f$, $\mathsf{op}(g) = \mathtt{write}(x,\_)$ and $g \xrightarrow{\mathsf{hb}_\mathcal{A}} e$, then $g \in \mathsf{PW}^x_\mathcal{A}(e)$. Hence it has to be $g \xrightarrow{\mathsf{hb}_\mathcal{A}} \max_{\mathsf{hb}_\mathcal{A}}(\mathsf{PW}^x_\mathcal{A}(e)) = f$. We have proved that there exist no $g \in E_\mathcal{A}$ such that $f \xrightarrow{\mathsf{hb}_\mathcal{A}} g \xrightarrow{\mathsf{hb}_\mathcal{A}} e$ and $\mathsf{op}(g) = \mathtt{write}(x,\_)$. Now it is straightforward to see that (Reads) is satisfied by $\mathcal{A}$.

Lemma 13. *Let $\mathcal{A} \in \mathsf{AbsPSI}$ be an abstract execution. For any $e, f \in E_\mathcal{A}$, $e \xrightarrow{\mathsf{rf}_\mathcal{A}} f$ if and only $\mathsf{op}(f) = \mathtt{read}(x,\_)$ and $e = \max_{\mathsf{hb}_\mathcal{A}}(\mathsf{PW}^x_\mathcal{A}(f))$ for some $x \in \mathsf{Obj}$.*

Proof. If $e \xrightarrow{\mathsf{rf}_\mathcal{A}} f$ then $e \xrightarrow{\mathsf{hb}_\mathcal{A}} f$, $\mathsf{op}(e) = \mathtt{write}(x,\_)$ and $\mathsf{op}(f) = \mathtt{read}(x,\_)$. The first three clauses give that $e \in \mathsf{PW}^x_\mathcal{A}(f)$. Also, whenever $g \in \mathsf{PW}^x_\mathcal{A}(f)$ and $g \neq e$, we have that $g \xrightarrow{\mathsf{hb}_\mathcal{A}} e$. This is because by definition of $\mathsf{PW}^x_\mathcal{A}(f)$, $\mathsf{op}(g) = \mathtt{write}(x,\_)$ and $g \xrightarrow{\mathsf{hb}_\mathcal{A}} f$. By (Wconflict) either $e \xrightarrow{\mathsf{hb}_\mathcal{A}} g$ or $g \xrightarrow{\mathsf{hb}_\mathcal{A}} e$. However, since $\mathsf{op}(g) = \mathtt{write}(x,\_)$, $g \xrightarrow{\mathsf{hb}_\mathcal{A}} f$ and $e \xrightarrow{\mathsf{rf}_\mathcal{A}} f$, the case $e \xrightarrow{\mathsf{hb}_\mathcal{A}} g$ leads to a contradiction, and therefore it has to be $g \xrightarrow{\mathsf{hb}_\mathcal{A}} e$. We have proved that $e \in \mathsf{PW}^x_\mathcal{A}(f)$, and whenever $g \in \mathsf{PW}^x_\mathcal{A}(f)$, $g \xrightarrow{\mathsf{hb}^*_\mathcal{A}} e$; that is, $e = \max_{\mathsf{hb}_\mathcal{A}}(\mathsf{PW}^x_\mathcal{A}(f))$.

Conversely, suppose that $e = \max_{\mathsf{hb}_\mathcal{A}}(\mathsf{PW}^x_\mathcal{A}(f))$, and $\mathsf{op}(f) = \mathtt{read}(x,\_)$. By definition of $\mathsf{PW}^x_\mathcal{A}(f)$ we have that $\mathsf{op}(e) = \mathtt{write}(x,\_)$, and $e \xrightarrow{\mathsf{hb}_\mathcal{A}} f$. Also, whenever $g \xrightarrow{\mathsf{hb}_\mathcal{A}} f$ for some g such that $\mathsf{op}(g) = \mathtt{write}(x,\_)$, we have that $g \in \mathsf{PW}^x_\mathcal{A}(f)$, hence $g \xrightarrow{\mathsf{hb}^*_\mathcal{A}} e$. That is, there exist no event g such that both $\mathsf{op}(g) = \mathtt{write}(x,\_)$, $g \xrightarrow{\mathsf{hb}_\mathcal{A}} f$ and $e \xrightarrow{\mathsf{hb}_\mathcal{A}} g$. By (Reads) and the definition of $\mathsf{rf}_\mathcal{A}$, it follows that $e \xrightarrow{\mathsf{rf}_\mathcal{A}} f$.

C PSI Concrete Executions

We assume a countable set of Replica Identifiers RId, whose elements are ranged over r, r' and a countable set of Transaction Identifiers TId, ranged over by $t, t', \cdots$. The set of concrete executions is defined formally by first modelling which events affect the state of the database in a geo-replicated system, then by defining the constraints that the transport layer of the distributed system has to satisfy.

Below we define formally what we mean by state of a (geo-replicated) database.

$$\text{(Start)}\ \frac{e \triangleright_{\mathcal{C}} t : \texttt{start} \ @\ r}{\langle L[r \mapsto \langle l, \varepsilon\rangle], M\rangle \xrightarrow{e}_{\mathcal{C}} \langle L[r \mapsto \langle l, (t, \texttt{start})\rangle], M\rangle}$$

$$\text{(Write)}\ \frac{e \triangleright_{\mathcal{C}} t : \texttt{write}(x, n) \ @\ r}{\langle L[r \mapsto \langle l, (t, \texttt{start}) \cdot l'\rangle], M\rangle \xrightarrow{e}_{\mathcal{C}} \langle L[r \mapsto \langle l, (t, \texttt{start}) \cdot l' \cdot (t, \texttt{write}(x, n))\rangle], M\rangle}$$

$$\text{(Read)}\ \frac{e \triangleright_{\mathcal{C}} t : \texttt{read}(x, n) \ @\ r \qquad \mathsf{val}(x, l \cdot (t, \texttt{start}) \cdot l') = n}{\langle L[r \mapsto \langle l, (t, \texttt{start}) \cdot l'\rangle], M\rangle \xrightarrow{e}_{\mathcal{C}} \langle L[r \mapsto \langle l, (t, \texttt{start}) \cdot l'\rangle], M\rangle}$$

$$\text{(Commit)}\ \frac{e \triangleright_{\mathcal{C}} (t : \texttt{commit} \ @\ r) \qquad \mathsf{canCommit}(r, L[r \mapsto \langle l, (t, \texttt{start}) \cdot l'\rangle])}{\langle L[r \mapsto \langle l, (t, \texttt{start}) \cdot l'\rangle], M\rangle \xrightarrow{e}_{\mathcal{C}} \langle L[r \mapsto \langle l \cdot (t, \texttt{start}) \cdot l', \varepsilon\rangle], M \cup \{(t, \texttt{start}) \cdot l'\}\rangle}$$

$$\text{(Abort)}\ \frac{e \triangleright_{\mathcal{C}} t : \texttt{abort} \ @\ r}{\langle L[r \mapsto \langle l, (t, \texttt{start}) \cdot l'\rangle], M\rangle \xrightarrow{e}_{\mathcal{C}} \langle L[r \mapsto \langle l, \varepsilon\rangle], M\rangle}$$

$$\text{(Receive)}\ \frac{e \triangleright_{\mathcal{C}} t : \texttt{receive}(m) \ @\ r \qquad m = (t, \texttt{start}) \cdot \_}{\langle L[r \mapsto \langle l, \varepsilon\rangle], M \cup \{m\}\rangle \xrightarrow{e}_{\mathcal{C}} \langle L[r \mapsto \langle l \cdot m, \varepsilon\rangle, M \cup \{m\}\rangle}$$

Fig. 11. Concrete Semantics of PSI

Definition 16 (Logs, States). *A* transaction log *is either an empty list, or a list of records of the form* $(t, \texttt{start}) :: l_t$*, where* $l \in \{(t, \texttt{write}(x, n)), (t, \texttt{read}(x, n)) \mid x \in \mathsf{Obj}, n \in \mathbb{N}\}^*$*; the set of all transaction logs is defined as* TrLog*. A* (Replica-local) Database Log *is obtained by concatenating multiple (possibly zero) transaction logs. The set of all database logs is denoted as* DbLog*. With an abuse of notation, we use* l, l' *to range both over transaction logs and database logs.*

A (replica)-local *state is a couple* $(l, l') \subseteq \mathsf{TrLog} \times \mathsf{DbLog}$.

A (database) state *is a pair of the form* (L, M)*, where* $L : \mathsf{repl} \to \mathsf{RState}$ *is a function, associating to each replica its local state (described below), and* $\mu \subseteq 2^{\mathsf{TrLog}}$ *is the set of messages in transit in the system. The set of all possible states is denoted as* State.

The initial state (L_0, M_0) *is the one in which the transaction and database logs at each replica are empty, and there are no messages in transit throughout the network:* $L_0 = \lambda r.(\varepsilon, \varepsilon), M_0 = \emptyset$.

Next, we give the definition of *Concrete Executions*. This definition is generic, and it does not assume that the database system is enforcing the PSI consistency level.

In Figure 11 we define a transition relation $\xrightarrow{e}$: $\mathsf{State} \times \mathsf{State}$, parametric in an event e and a concrete execution $\mathcal{C}$, which defines which events can take place in a given state of a database enforcing the PSI consistency level. The rules of Figure 11 are the formal counterpart to the pseudo code given in Figure 2. In Rule `(Read)`, the function $\mathsf{val}(x, l)$ returns the value n of the last record of the form $(t, \texttt{write}(x, n))$ which appears in l, 0

$$(e \triangleright t : \texttt{start} \ @ \ \_) \wedge (f \triangleright t : \texttt{start} \ @ \ \_) \implies e = f \qquad \text{(OneStart)}$$

$$(\_ \triangleright t : \_ \ @ \ \_) \implies \exists f.\ (f \triangleright t : \texttt{commit} \ @ \ \_) \vee (f \triangleright t : \texttt{abort} \ @ \ \_) \qquad \text{(EndTr)}$$

$$\_ \triangleright (t : \texttt{commit} \ @ \ r) \implies \neg(\_ \triangleright t : \texttt{receive}(\_) \ @ \ r) \qquad \text{(NoSameRcv)}$$

$$(e \triangleright t : \texttt{receive}(\_) \ @ \ r) \wedge (f \triangleright t : \texttt{receive}(\_) \ @ \ r) \implies e = f \qquad \text{(OneRcv)}$$

$$\begin{aligned}(\_ \triangleright t : \{\texttt{receive}(\_), \texttt{commit}\} \ @ \ r) < (\_ \triangleright t' : \texttt{start} \ @ \ r) \wedge (r \neq r') \\ \implies (\_ \triangleright t : \texttt{receive}(\_) \ @ \ r') < (\_ \triangleright t' : \texttt{receive}(\_) \ @ \ r')\end{aligned} \qquad \text{(CDel)}$$

Fig. 12. Transport Layer and Database Requirements: all free variables are universally quantified

if no such record exists. In Rule `(Commit)`, the predicate $\mathsf{canCommit}(r, l)$ is defined as

$$\begin{aligned}\mathsf{canCommit}(r, L) \iff \big(\forall x, t.\ (L(r) = (\_,\ \ (\_ \cdot (t, \texttt{write}(x, \_)) \cdot \_)) \implies \\ \forall r', t'.\ (L(r') = ((\_ \cdot (t', \texttt{write}(x, \_)) \cdot \_), \_) \implies \\ L(r) = ((\_ \cdot (t', \texttt{write}(x, \_)) \cdot \_), \_))\big)\end{aligned}$$

The rules of the operational semantics alone are not sufficient to define the set of all concrete executions which can be produced by a database which behaves according to the PSI specification given in §2. It is also necessary to define the *Transport Layer* and the *Database Requirements* of the distributed system; these are the set of rules that regulate the deliveries of messages broadcast abroad replicas, and the set of rules that the database uses to assign transaction identifiers. Their definition is given in Figure 12. A transaction identifier is never assigned to multiple transactions (OneStart), a transaction either commits or abort (EndTr). A message is never delivered to the replica that originally broadcast it (NoSameRcv), and it is received at most one by other replicas (OneRcv). Also, sequential transactions appear to commit in the same order at each replica (CDel).

We are now in position to define the set of PSI concrete executions. These are the (possibly infinite) sequences of events that can be generated by an initial state of the system which respect the requirements of Figure 12.

Definition 17 (PSI Concrete Executions). *A concrete execution $\mathcal{C}$ is* PSI *if $E_{\mathcal{C}} = \{e_0, e_1, \cdots\}$, $\prec_{\mathcal{C}} = \{(e_i, e_{i+1}) \mid i \geq 0\}$, there exists a set of states $\{(L_i, M_i)\}_{i \geq 0}$ where (L_0, M_0) is the initial state of the system, for any $i \geq 0$, $(L_i, M_i) \xrightarrow{e_i}_{\mathcal{C}} (L_{i+1}, M_{i+1})$ and $\mathcal{C}$ satisfies the equations of Figure 12.*

The set of PSI concrete executions is denoted as $\mathsf{ConcPSI}$.

D Proof of Theorem 1

The proof of Theorem 1 is performed by proving that $\mathsf{history}(\mathsf{ConcPSI})$ is included in $\{\mathcal{H} \in \mathsf{Hist} \mid \exists \mathsf{hb}.(\mathcal{H}, \mathsf{hb}) \in \mathsf{AbsPSI}\}$, and vice versa. For the moment, we focus on proving the following inclusion:

Theorem 3 (Soundness).

$$\mathsf{history}(\mathsf{ConcPSI}) \subseteq \{\mathcal{H} \in \mathsf{Hist} \mid \exists \mathsf{hb}.(\mathcal{H}, \mathsf{hb}) \in \mathsf{AbsPSI}\}$$

Proof. Let us fix an execution $\mathcal{C} \in \mathsf{ConcPSI}$. Let $\mathcal{H} = \mathsf{history}(\mathcal{C})$; recall that the definition of the single components of $\mathcal{H}$ is given in Definition 5, so that it remains to define a suitable happens before relation hb such that $(\mathcal{H}, \mathsf{hb}) \in \mathsf{AbsPSI}$. The definition of such a relation has been already given in §4, and we repeat it here for the sake of completeness. Let

$$\begin{aligned}\mathsf{hb}_{\mathcal{A}} = \mathsf{co}_{\mathcal{A}} \cup \{(e,f) \in E_{\mathcal{A}} \times E_{\mathcal{A}} \mid \exists r, r', t, t', g.\, (r \neq r') \wedge (e \rhd_{\mathcal{C}} t : \_ \mathbin{@} r) \wedge \\ (f \rhd_{\mathcal{C}} t' : \_ \mathbin{@} r') \wedge \quad (g \rhd_{\mathcal{C}} t : \mathtt{receive}(\_) \mathbin{@} r') \wedge (g \prec_{\mathcal{C}} f)\}.\end{aligned}$$

Note that the operational semantics and the transport layer constraints (Figure 12) ensure that **(i)** transactions at a given replica are executed sequentially, **(ii)** a message is never delivered at a replica r' while it is executing some transaction, **(iii)** a message delivered at replica r' consists of the transaction log of some transaction t that committed at some other replica r, and **(iv)** an operation performed at some transaction t can be performed at replica r only if transaction t started at such a replica. These properties ensure that whenever $(e \rhd_{\mathcal{C}} t : \_ \mathbin{@} r) \prec_{\mathcal{C}} (f \rhd_{\mathcal{C}} t' : \_ \mathbin{@} r)$, then either $t = t'$, or $(\_ \rhd t : \mathtt{commit} \mathbin{@} r) \prec_{\mathcal{C}} (\_ \rhd t' : \mathtt{start} \mathbin{@} r)$. They also ensure that whenever $r \neq r'$, and $e \prec_{\mathcal{C}} f$, then $(\_ \rhd t : \mathtt{receive}(\_) \mathbin{@} r' \prec_{\mathcal{C}} \_ \rhd t' : \mathtt{start} \mathbin{@} r')$.

It is now easy to prove that hb is both transitive and irreflexive. It is irreflexive because $\mathsf{hb} \subseteq \prec_{\mathcal{C}}$, and the latter is also irreflexive Proving that hb is transitive is less obvious; suppose that $e \xrightarrow{\mathsf{hb}} f \xrightarrow{\mathsf{hb}} g$. We perform a case analysis on why $e \xrightarrow{\mathsf{hb}} f$ and $e \xrightarrow{\mathsf{hb}} g$; we give details only for the most interesting case, which corresponds to $e \rhd t : \_ \mathbin{@} r$, $f \rhd t' : \_ \mathbin{@} r'$, $g \rhd t'' : \_ \mathbin{@} r''$, where t, t', t'' are distinct, and so are r, r'. In this case we have that $(\_ \rhd t : \mathtt{receive}(\_) \mathbin{@} r') \prec_{\mathcal{C}} (\_ \rhd t' : \mathtt{start} \mathbin{@} r')$. The rules of the operational semantics also ensure that $(\_ \rhd t : \mathtt{commit} \mathbin{@} r) \prec_{\mathcal{C}} (\_ \rhd t : \mathtt{receive}(\_) \mathbin{@} r')$. Next, we consider two different sub cases, according to whether $r = r''$ or $r \neq r''$. If $r = r''$, then by hypothesis we have that $r' \neq r''$. Since $f \xrightarrow{\mathsf{hb}} g$, $\mathsf{repl}_{\mathcal{H}}(f) = r', \mathsf{repl}_{\mathcal{H}}(g) = r$, then by definition of hb we obtain that $(\_ \rhd t' : \mathtt{receive}(\_) \mathbin{@} r) \prec_{\mathcal{C}} (\_ \rhd t'' : \mathtt{start} \mathbin{@} r)$; the operational semantics also ensures that $(\_ \rhd t' : \mathtt{start} \mathbin{@} r') \prec_{\mathcal{C}} (\_ \rhd t' : \mathtt{receive}(\_) \mathbin{@} r)$. Now we have that

$$\begin{aligned}(\_ \rhd t : \mathtt{commit} \mathbin{@} r) \prec_{\mathcal{C}} (\_ \rhd t : \mathtt{receive}(\_) \mathbin{@} r') \prec_{\mathcal{C}} (\_ \rhd t' : \mathtt{start} \mathbin{@} r') \prec_{\mathcal{C}} \\ (\_ \rhd t' : \mathtt{receive}(\_) \mathbin{@} r) \prec_{\mathcal{C}} (\_ \rhd t'' : \mathtt{start} \mathbin{@} r'')\end{aligned}$$

Since $\mathsf{repl}_{\mathcal{H}}(e) = \mathsf{repl}_{\mathcal{H}}(g) = r$, the definition of co now gives that $e \xrightarrow{\mathsf{co}_{\mathcal{H}}} g$, and the definition of hb leads to $e \xrightarrow{\mathsf{hb}} g$. If $r \neq r''$, then either $r' = r''$ or $r' \neq r''$. We only consider this last case. Since $\mathsf{repl}_{\mathcal{H}}(f) = r', \mathsf{repl}_{\mathcal{H}}(g) = r''$, in this case we have that $f \xrightarrow{\mathsf{hb}} g$ because $(\_ \rhd t' : \mathtt{receive}(\_) \mathbin{@} r'') \prec_{\mathcal{C}} (\_ \rhd t'' : \mathtt{start} \mathbin{@} r'')$. Also, since $(\_ \rhd t : \mathtt{receive}(\_) \mathbin{@} r') \prec_{\mathcal{C}} (\_ \rhd t' : \mathtt{start} \mathbin{@} r')$, by Property (CDel) we have that $(\_ \rhd t : \mathtt{receive}(\_) \mathbin{@} r'') \prec_{\mathcal{C}} (\_ \rhd t' : \mathtt{receive}(\_) \mathbin{@} r'')$. As a straightforward consequence, we obtain that $(\_ \rhd t : \mathtt{commit} \mathbin{@} r) \prec_{\mathcal{C}} (\_ \rhd t : \mathtt{receive}(\_) \mathbin{@} r'') \prec_{\mathcal{C}}$

$(\_ \triangleright t'' : \texttt{start} \ @ \ r'')$. Since $\mathsf{repl}_{\mathcal{H}}(e) = r, \mathsf{repl}_{\mathcal{H}}(g) = r''$, the definition of hb ensures that $e \xrightarrow{\mathsf{hb}} g$.

Proving that $\sim$ is an equivalence relation is also obvious, for whenever $(e \triangleright_{\mathcal{C}} t : \_ \ @ \ \_) \in E_{\mathcal{H}}$ then $[e]_{\sim} = \{f \mid (f \triangleright_{\mathcal{C}} t : \_ \ @ \ \_\}$ so that for all $e, f \in E_{\mathcal{H}}$ either $[e]_{\sim} = [f]_{\sim}$ or $[e]_{\sim} \cap [f]_{\sim} = \emptyset$. It is also contiguous with respect to $\mathsf{co}_{\mathcal{H}}$; suppose in fact that $e \xrightarrow{\mathsf{co}} f \xrightarrow{\mathsf{co}_{\mathcal{H}}} g$, and $e \sim g$. According to the definition of $\sim$ and $\mathsf{co}_{\mathcal{H}}$, $(e \triangleright t : \_ \ @ \ r)$, $(f \triangleright t' : \_ \ @ \ r)$, and $(g \triangleright t : \_ \ @ \ r)$. The operational semantics, and (OneStart), ensure then that $t = t'$, from which it follows that $e \sim f$. Finally, each equivalence class $[e]_{\sim}$ is finite, because of (EndTr).

Let us turn our attention to the axioms of the PSI consistency level, Figure 4.

- (Chains)**:** this Axiom is trivially satisfied because of the definition of hb.
- (Atomic)**:** Suppose then that $e, fe', f' \in E_{\mathcal{H}}$ are some events such that $e \sim e', f \sim f'$ and $e' \xrightarrow{\mathsf{hb}} f'$. Assume also that $e' \not\sim f'$, and that $(e' \triangleright t : \_ \ @ \ r)$, $(f' \triangleright t' : \_ \ @ \ r')$. By definition of $\sim$, we have that $t \neq t'$, $(e \triangleright_{\mathcal{C}} t : \_ \ @ \ r)$, and $(f \triangleright_{\mathcal{C}} t' : \_ \ @ \ r')$. There are two possible cases: either $r = r'$, in which case $e' \xrightarrow{\mathsf{hb}} f'$ implies that $(\_ \triangleright_{\mathcal{C}} t : \texttt{commit} \ @ \ r) \prec_{\mathcal{C}} (\_ \triangleright t' : \texttt{start} \ @ \ r')$, from which it follows that $e \xrightarrow{|hb} f$; or $r \neq r'$, in which case $e' \xrightarrow{\mathsf{hb}} f'$ implies that $(\_ \triangleright_{\mathcal{C}} t : \texttt{receive}(\_) \ @ \ r') \prec_{\mathcal{C}} (\_ \triangleright_{\mathcal{C}} t' : \texttt{start} \ @ \ r')$. Therefore, (Atomic) is satisfied.
- (Wconflict)**:** Let $e, f \in E_{\mathcal{A}}$ be two events such that $\mathsf{op}_{\mathcal{H}}(e) = \texttt{write}(x, \_)$, $\mathsf{op}_{\mathcal{H}}(f) = \texttt{write}(x, \_)$ and $e \neq f$. Then either $e \prec_{\mathcal{C}} f$ or $f \prec_{\mathcal{C}} e$. We only consider the first case, as the second one is symmetric; note that if we assume that $\mathsf{repl}_{\mathcal{H}}(e) = \mathsf{repl}_{\mathcal{H}}(f)$, then by definition $e \xrightarrow{\mathsf{co}_{\mathcal{H}}} f$, which implies $e \xrightarrow{\mathsf{hb}} f$. Therefore we can assume that $\mathsf{repl}_{\mathcal{H}}(e) \neq \mathsf{repl}_{\mathcal{H}}(f)$, which in turn gives that $\mathsf{trans}_{\mathcal{H}}(e) \neq \mathsf{trans}_{\mathcal{H}}(f)$. By the hypothesis that $e \prec_{\mathcal{C}} f$ we have that $(e \triangleright_{\mathcal{C}} t : \texttt{write}(x, \_) \ @ \ r)$, $(f \triangleright t'_{\mathcal{C}} : \texttt{write}(x, \_) \ @ \ r'$. The fact that both $e, f \in E_{\mathcal{H}}$ ensures that $(\_ \triangleright t : \texttt{commit} \ @ \ r), (\_ \triangleright t' : \texttt{commit} \ @ \ r') \in E_{\mathcal{C}}$. However, this is possible only if $(\_ \triangleright_{\mathcal{C}} t : \texttt{receive} \ @ \ r') \prec_{\mathcal{C}} (\_ \triangleright_{\mathcal{C}} t' : \texttt{commit} \ @ \ r')$, since the predicate $\mathsf{canCommit}(r', L)$ has to be satisfied in the global state $(L, \_)$ in which the event $(\_ \triangleright_{\mathcal{C}} t' : \texttt{commit} \ @ \ r')$ was performed. The rules of the operational semantics ensure that the delivery of the effects of t at replica r' also cannot happen while a transaction is executing at such a replica. Hence we also have that $(\_ \triangleright t : \texttt{receive} \ @ \ r') \prec_{\mathcal{C}} (\_ \triangleright t' : \texttt{start} \ @ \ r')$, which in turn gives $e \xrightarrow{\mathsf{hb}} f$.
- (Reads)**:** Since we have already proved that $(\mathcal{H}, \mathsf{hb})$ satisfies (Wconflict), we can employ Lemma 12 so that it suffices to prove that (RFmax) is satisfied. Since $\mathcal{C} \in \mathsf{ConcPSI}$ by hypothesis, we know that $E_{\mathcal{H}} = \{e_0, e_1, \cdots\}$, $\prec_{\mathcal{C}} = \{(e_i, e_{i+1}) \mid i \geq 0\}^+$, and there exist $\{(L_i, M_i)\}_{i \geq 0}$ such that (L_0, M_0) is the initial state and, for any $i \geq 0$, $(L_i, M_i) \xrightarrow{e_i}_{\mathcal{C}} (L_{i+1}, M_{i+1})$. Let then $i \geq 0$, and assume that $\mathsf{op}_{\mathcal{H}}(x) = \texttt{read}(x, n)$. That is, $(e_i \triangleright_{\mathcal{C}} t : \texttt{read}(x, n) \ @ \ r)$ for some t, r. By definition, we have

$$\mathsf{PW}^{x}_{(\mathcal{H}, \mathsf{hb})}(e_i) = \{f \mid \exists t, r. (f \triangleright_{\mathcal{C}} t' : \texttt{write}(x, \_) \ @ \ r') \wedge$$
$$\big((r = r' \wedge f \prec_{\mathcal{C}} e) \vee (r \neq r' \wedge (\_ \triangleright t' : \texttt{receive}(\_) \ @ \ r \prec_{\mathcal{C}} (\_ \triangleright t : \texttt{start} \ @ \ r))\big)\}$$

  According to Lemma 11 there are two possible cases. The first one is $\mathsf{PW}^{x}_{(\mathcal{H}, \mathsf{hb})}(e_i) = \emptyset$, in which case $L_i(r) = (l, (t, \texttt{start}))$ for some l which does

not contain any write record of the form $\texttt{write}(x, \_)$, hence the operational semantics ensure that $n = 0$.
In the second case we have that $\max_{\mathsf{hb}}(\mathsf{PW}^x_{(\mathcal{H},\mathsf{hb})}(e_i)) = f$ for some f such that $\mathsf{op}(f) = \texttt{write}(x, m)$. From the point of view of the concrete execution, we have that $(f \triangleright_{\mathcal{C}} t' : \texttt{write}(x, m) \mathbin{@} r')$ in this case we have to perform a case analysis on whether $r = r'$ or $r \neq r'$. We restrict our attention to this latter case, as it is the most difficult one. The definition of hb and $\max$ ensure that in $\mathcal{C}$ the event f is the last write to object x in transaction t', prior to the execution of event e_i, or it is the last write to object x in transaction t', $(\_ \triangleright_{\mathcal{C}} t'\texttt{receive}(l_{t'}) \mathbin{@} r) \prec_{\mathcal{C}} (\_ \triangleright_{\mathcal{C}} t : \texttt{start} \mathbin{@} r)$, and whenever $(\_ \triangleright_{\mathcal{C}} t''\texttt{receive}(l_{t''}) \mathbin{@} r) \prec_{\mathcal{C}} (\_ \triangleright_{\mathcal{C}} t : \texttt{start} \mathbin{@} r)$ then either $(\_ \triangleright_{\mathcal{C}} t''\texttt{receive}(l_{t''}) \mathbin{@} r) \prec_{\mathcal{C}} (\_ \triangleright_{\mathcal{C}} t'\texttt{receive}(l_{t'}) \mathbin{@} r)$ or $l_{t''}$ does not contain any record of the form $(t'', \texttt{write}(x, \_))$. Also, there exists no event g such that $g \triangleright_{\mathcal{C}} t : \texttt{write}(x, \_) \mathbin{@} r$ and $g \prec_{\mathcal{C}} e$.
Now it is not difficult to note that $L_i(r) = ((l \cdot l_{t'} \cdot l'), l'')$ for some database log l and two other database logs l', l'' which do not contain any record of the form $\texttt{write}(x, \_)$. According to the operational semantics, the value n read in event e coincides with the last write to object x in $l \cdot l_{t'} \cdot l' \cdot l''$. This is exactly the last write record over object x in $l_{t'}$, which has the form $(t', \texttt{write}(x, m))$ since the last event writing to object x in t' is f, and $\mathsf{op}_{\mathcal{C}})(f) = \texttt{write}(x, m)$. Therefore, $m = n$.

Next, we focus on proving the opposite inclusion of Theorem 3, that is

Theorem 4.

$$\{\mathcal{H} \in \mathsf{Hist} \mid \exists \mathsf{hb}.(\mathcal{H}, \mathsf{hb}) \in \mathsf{AbsPSI}\} \subseteq \mathsf{history}(\mathsf{ConcPSI})$$

Proof. We have to define, for each abstract execution $\mathcal{A} = (\mathcal{H}, \mathsf{hb}) \in \mathsf{AbsPSI}$, a concrete execution $\mathsf{Concrete}(\mathcal{A}) \in \mathsf{ConcPSI}$ such that $\mathsf{history}(\mathsf{Concrete}(\mathcal{A})) = \mathcal{H}$. Before supplying the formal details of the proof, it is mandatory to define formally how $\mathsf{Concrete}(\mathcal{A})$ is defined.

First, let $\mathcal{A} = (\mathcal{H}, \mathsf{hb})$ be an abstract execution in AbsPSI. We assume an enumeration $\mathsf{enum} : \mathsf{TId} \to \mathbb{N}$, and we say that $t < t'$ whenever $\mathsf{enum}(t) < \mathsf{enum}(t')$. The function enum exists because we are assuming that TId is countable. With an abuse of notation, we write $t + 1$ to denote the immediate successor of t according to enum.

We write $t_1 \lhd_{\mathsf{hb}} t_2$ whenever $t_1 = \mathsf{TIdOf}([e]_\sim), t_2 = \mathsf{TIdOf}([f]_\sim)$ for some$[e]_\sim, [f]_\sim$ such that $[e]_\sim \xrightarrow{(\mathsf{hb}\setminus\sim)_{/\sim}} [f]_\sim$, and there exists no $[g]_\sim$ such that $[e]_\sim \xrightarrow{(\mathsf{hb}\setminus\sim)_{/\sim}} [g]_\sim \xrightarrow{(\mathsf{hb}\setminus\sim)_{/\sim}} [f]_\sim$.

We assume the existence of a one-to-one mapping $\mathsf{TIdOf} : E_{\mathcal{H}/\sim} \to T$, such that whenever $[e]_\sim \xrightarrow{\mathsf{hb}/\sim} [f]_\sim$ then $\mathsf{TIdOf}([e]_\sim \leq \mathsf{TIdOf}([f]_\sim)$. We also assume that TIdOf is contiguous, meaning that whenever $t, t' \in \mathsf{img}(\mathsf{TIdOf})$ and $t \leq t'' \leq t'$, then $t'' \in \mathsf{img}(\mathsf{TIdOf})$. This mapping is ensured to exist if we assume that hb is acyclic

Finally, we assume a one-to-one mapping $\mathsf{ReplOf} : E_{\mathcal{H}/\approx} \to \mathsf{RId}$, which maps chains to replicas. This mapping trivially exists, as RId is assumed to be infinite.

For any $t \in \mathsf{img}(\mathsf{TIdOf})$, we will often need to identify which replica is running such a transaction.Let $\mathsf{TRtoCH} : E_{/\sim} \to E_{/\approx}$ be defined as$\mathsf{TRtoCH}([e]_\sim) = [e]_\approx$; this

function maps each transaction to the chain which executes such a transaction. Then, the replica which executes transaction t is given by $\mathsf{ReplOf}(\mathsf{TRtoCH}(\mathsf{TIdOf}^{-1}(t))$. Since we will need to use this function often, we will just denote it as $\mathsf{ReplOfTrans}(t)$.

Finally, given $[e]_\sim \in E_{\mathcal{H}/\sim}$, we define the transaction log $\mathsf{LogOf}([e]_\sim) = (\mathsf{TIdOf}([e]_\sim), \mathtt{start}) \cdot l_{[e]_\sim}$, where for any $F \subseteq [e]_\sim$ the set l_F is defined as follows:

- $l_F = \varepsilon$ if $F = \emptyset$,
- $l_F = l_{F \setminus \{\min_{\mathsf{co}}(F)\}}$ if $\mathsf{op}(\min_{\mathsf{co}}(F)) = \mathtt{read}(\_)$,
- $L_f = (\mathsf{TIdOf}([e]_\sim), \mathtt{write}(x, n)) \cdot l_{F \setminus \{\min_{\mathsf{co}}(F)\}}$ if $\mathsf{op}(\min_{\mathsf{co}}(F)) = \mathtt{write}(x, n)$.

For any $t \in \mathsf{img}(\mathsf{TIdOf})$, we let $\mathsf{LogOfTrans}(t) = \mathsf{LogOf}(\mathsf{TIdOf}^{-1}(t))$.

The set $E_{\mathsf{Concrete}(\mathcal{A})}$ is defined as $E_{\mathtt{start}} \uplus E_{\mathcal{H}} \uplus E_{\mathtt{commit}} \uplus E_{\mathtt{receive}}$, where:

- $E_{\mathtt{start}} = \{e_{\mathtt{start}(t)} \mid t \in \mathsf{img}(\mathsf{TIdOf})\}$,
- $E_{\mathtt{commit}} = \{e_{\mathtt{commit}(t)} \mid t \in \mathsf{img}(\mathsf{TIdOf})\}$,
- $E_{\mathtt{receive}} = \{e_{\mathtt{receive}(t1,t2)} \mid (t_1 \lhd_{\mathsf{hb}^*} t_2 \wedge \mathsf{ReplOfTrans}(t_1) \neq \mathsf{ReplOfTrans}(t_2)) \wedge (\forall t'.t' < t_2 \implies \mathsf{ReplOfTrans}(t') = \mathsf{ReplOfTrans}(t_2) \implies \neg(t_1 \lhd^*_{\mathsf{hb}} t'))\}$

Let us now turn our attention to the attributes $\mathsf{repl}_{\mathsf{Concrete}(\mathcal{A})}$, $\mathsf{trans}_{\mathsf{Concrete}(\mathcal{A})}$ and $\mathsf{op}_{\mathsf{Concrete}(\mathcal{A})}$, which are defined simultaneously. For $e_{\mathtt{start}(t)} \in E_{\mathtt{start}}$, we let

$$(e_{\mathtt{start}(t)} \rhd_{\mathsf{Concrete}(\mathcal{A})} t : \mathtt{start} \mathbin{@} \mathsf{ReplOfTrans}(t))$$

and similarly for $e_{\mathtt{commit}(t)} \in E_{\mathtt{commit}}$, we let

$$(e_{\mathtt{commit}(t)} \rhd_{\mathsf{Concrete}(\mathcal{A})} t : \mathtt{commit} \mathbin{@} \mathsf{ReplOfTrans}(t))$$

For any $e_{\mathtt{receive}(t_1,t_2)} \in E_{\mathtt{receive}}$, we let

$$(e_{\mathtt{receive}(t_1,t_2)} \rhd_{\mathsf{Concrete}(\mathcal{A})} t_1 : \mathtt{receive}(\mathsf{LogOfTrans}(t_1)) \mathbin{@} \mathsf{ReplOfTrans}(t_2))$$

Finally, for any $e \in E_{\mathcal{H}}$ we let

$$(e \rhd_{\mathsf{Concrete}(\mathcal{A})} = \mathsf{TIdOf}([e]_\sim) : \mathsf{op}_{\mathcal{H}}(e) \mathbin{@} \mathsf{ReplOf}([e]_\approx))$$

It remains to define the execution order of events; in practice, we define a computation γ of the form $(L_0, M_0) \xrightarrow{e_0}_{\mathsf{Concrete}(\mathcal{A})} (L_1, M_1) \xrightarrow{e_1}_{\mathsf{Concrete}(\mathcal{A})} \cdots$, from which the execution order $\prec_{\mathsf{Concrete}(\mathcal{A})}$ can be easily derived. First, for any transaction $[e]_\sim \in E_{\mathcal{H}/\sim}$ we define the state $(L_{[e]_\sim}, M_{[e]_\sim})$ as follows:

- for any replica $r \in \mathsf{RId}$ and transaction identifier $t \in \mathsf{TId}$, let $\mathsf{LastTransOf}(r, t) = \max_<(t' \mid \mathsf{ReplOf}(t') = r \wedge t' \leq t\}$; then given $r \in \mathsf{RId}$, we let $L_{[e]_\sim}(r) = (\varepsilon, l_r)$, where $l_r = \mathsf{LogOfTrans}(t_1) \cdot \mathsf{LogOfTrans}(t_2) \cdot \cdots \cdot \mathsf{LogOfTrans}(t_n)$, where $t_n = \mathsf{LastTransOf}(r, \mathsf{TIdOf}([e]_\sim), t_1 < \cdots < t_n$ and $\{t \mid t \lhd_{\mathsf{hb}} t_n\} = \{t_i\}_{i=1}^n$,
- $M_{[e]_\sim} = \{\mathsf{LogOf}(t) \mid t < \mathsf{TIdOf}([e]_\sim)\}$.

For any $t \in \mathsf{img}(\mathsf{TIdOf})$, we now define a computation fragment γ_t. Let $[e]_\sim$ be such that $t = \mathsf{TIdOf}([e]_\sim)$, and $L_{[e]_\sim}$ be defined as above. Suppose also that $[e]_\sim = \{e_i\}_{i=1}^n$, and $e_i \xrightarrow{\mathsf{co}_\mathcal{A}} e_{i+1}$; We let

$$\gamma_{\mathsf{exec}}(t) = (L_{[e]_\sim}, M_{[}e]_\sim) \xrightarrow{e_{\mathtt{start}(t)}} (L^0_{[e]_\sim}, M_{[e]_\sim}) \xrightarrow{e_0} \cdots$$
$$\cdots \xrightarrow{e_n} (L^n_{[e]_\sim}, M_{[e]_\sim}) \xrightarrow{e_{\mathtt{commit}}} L^{\mathsf{final}}_{[e]_\sim}, M^{\mathsf{final}}_{[e]_\sim})$$

where the global states above can be defined according to the operational semantics of Figure 11. Note that this gives $M^{\mathsf{final}}_{[e]_\sim} = M_{[e]_\sim} \cup \{\mathsf{LogOf}([e]_\sim\}$, which is exactly $M_{[f]_\sim}$ for $[f]_\sim = \mathsf{TIdOf}^{-1}(t+1)$.

Also, for any t such that $t+1 \in \mathsf{img}(\mathsf{TIdOf})$, we define the computation fragment $\gamma_{\mathsf{idle}}(t+1)$ as follows: if $\mathsf{ReplOfTrans}(t) = \mathsf{ReplOfTrans}(t')$, then $\gamma_{\mathsf{idle}}(t+1)$ is defined to be the computation fragment with no transitions $(L^{\mathsf{final}}_{[e]_\sim}, M^{\mathsf{final}}_{[e]_\sim})$, where $[e]_\sim = \mathsf{TIdOf}^{-1}(t)$, $[f]_\sim = \mathsf{TIdOf}^{-1}(t+1)$. Otherwise, let also $\{t_i\}_{i=1}^n = \{e_{\mathtt{receive}(t',t+1)} \in E_{\mathtt{receive}} \mid t' < t+1\}$; then $\gamma_{\mathsf{idle}}(t+1)$ is defined as

$$(L^{\mathsf{final}}_{[e]_\sim}, M^{\mathsf{final}}_{[e]_\sim}) \xrightarrow{e_1} \cdots \xrightarrow{e_n} (L_{[f]_\sim}, M^{\mathsf{final}}_{[e]_\sim})$$

where $[f]_\sim = \mathsf{TIdOf}^{-1}(t+1)$, and the intermediate states of the computation fragment are determined uniquely, according to Rule `(Receive)` of the operational semantics.

Finally, given two (finite) computation fragments γ, γ', let us define $\gamma \cdot \gamma'$ to be the concatenation of the γ and γ', provided that the last state of γ coincides with the first state of γ'. Then, let $\mathsf{img}(\mathsf{TIdOf}) = \{t_n\}_{n\in\mathbb{N}}$, and assume that for any $i, j \geq 0$, $t_i \leq t_{i+j}$; the computation γ is defined by letting

$$\gamma = \gamma_{\mathsf{exec}}(t_0)\gamma_{\mathsf{idle}}(t_1) \cdot \gamma_{\mathsf{exec}(t_1} \cdot \gamma_{\mathsf{idle}}(t_2) \cdots$$

Next, we prove that the execution γ can be generated by the rules of the operational semantics. Given a transition $(L, M) \xrightarrow{e} (L', M')$, we perform a case analysis on which rule of the operational semantics has been applied to derive the transition above. There are only two non trivial cases to consider.

- $(L, M) \xrightarrow{e} (L', M')$ because of Rule `(Read)`. In this case, we know that $\mathsf{op}_\mathcal{H}(e) = \mathtt{read}(x, n)$, for some x, n, and this judgement belongs to $\gamma_{\mathsf{exec}}(t_i)$ for some transaction identifier t_i; also, $\mathsf{TIdOf}([e]_\sim) = t_i$.
  Let $L(r) = (l, l')$; we need to check that **(i)** $l' = (t_i, \mathtt{start}) \cdot l''$ for some l'', and **(ii)** $\mathsf{val}(x, l \cdot l') = n$. Statement **(i)** follows as an easy consequence of the Definition of $\gamma_{\mathsf{exec}}(t_i)$, whose first transition is inferred via Rule `(Start)`, and from the fact that $\mathsf{TIdOf}([e]_\sim) = t_i$.
  For **(ii)**, since $\mathcal{A} \in \mathsf{AbsPSI}$, we can employ Lemma 12and infer that either **(a)** $n = 0$ and $\mathsf{PW}^x_{\mathsf{AbsPSI}}(e) = \emptyset$, or **(b)** $\mathsf{op}(\max_{\mathsf{hb}}(\mathsf{PW}^x_{\mathsf{AbsPSI}}(e)) = \mathtt{write}(x, n)$.
  In case **(a)**, this means that there exists no event f such that $\mathsf{op}(f) = \mathtt{write}(x, \_)$ and $f \xrightarrow{\mathsf{co}} e$, and no transaction $[g]_\sim$ such that $[g]_\sim \xrightarrow{\mathsf{hb}/\sim} [e]_\sim$ and $\mathsf{op}(f) = \mathtt{write}(x, \_)$ for some $f \in [g]_\sim$. Therefore, in $\gamma_{(\mathsf{exec})}(t_i)$ there are no transitions of the form $(L_a, M_a) \xrightarrow{f} (L_b, M_b)$ which precede $(L, M) \xrightarrow{e} (L', M')$, and such

that $\mathsf{op}_{\mathsf{Concrete}(\mathcal{A})} = \mathtt{write}(x, \_)$. A trivial consequence of this fact is that l' contains no record of the form $(t_i, \mathtt{write}(x, \_))$. Also, whenever t_j is such that $t_j \lhd_{\mathsf{hb}}^* t_i$, then there exists no record of the form $(t_j, \mathtt{write}(x, \_))$ which belongs to the transaction log $\mathsf{LogOfTrans}(t_j)$. By definition, then no record of the form $(\_, \mathtt{write}(x, \_))$ belongs to l, and in particular to$l \cdot l'$; also by definition, $\mathsf{val}(x, l \cdot l') = 0$.
In case **(b)**, let $f := \max_{\mathsf{hb}}(\mathsf{PW}_{\mathcal{A}}^x(e)$, and assume that $\mathsf{op}_{\mathcal{H}}(f) = \mathtt{write}(x, n)$. If $\mathsf{TIdOf}([f]_\sim = \mathsf{TIdOf}([e]_\sim) = t_i$, then the definition of $\gamma_{\mathsf{exec}}(t_i)$ ensures that there exists one record of the form $(t_i, \mathtt{write}(x, m))$ in l'; furthermore, it is the last record of the form $(\_, \mathtt{write}(x, \_))$ which appears in l'. Therefore, $\mathsf{val}(x, l \cdot l') = n$.
If $\mathsf{TIdOf}([f]_\sim) \neq \mathsf{TIdOf}([e]_\sim)$, then there exists no record of the form $(\_\mathtt{write}(x, \_))$ which belongs to l'. Also, by (Wconflict), we know that the set of transactions which write to object x and which precedes $[e]_\sim$, according to $\mathsf{hb}_{/\sim}$, is totally ordered. In particular, the set of transaction identifiers

$$\{t_j \mid t_j \lhd_{\mathsf{hb}} t_i \wedge (t_j, \mathtt{write}(x, \_)) \text{ appears in } \mathsf{LogOfTrans}(t_j)\}$$

is totally ordered according to $\lhd_{\mathsf{hb}}$, Let $t^1 \lhd_{\mathsf{hb}} t^2 \lhd_{\mathsf{hb}} \cdots \lhd_{\mathsf{hb}} t^h$ be such a total order; we have that

$$\mathsf{val}(x, \mathsf{LogOfTrans}(t^1) \cdot \cdots \cdot \mathsf{LogOfTrans}(t^h)) = n$$

Finally, remember that we constructed the function $\mathsf{TIdOf}(\cdot)$ so that whenever $t \lhd_{\mathsf{hb}} t'$. Therefore, the order of the logs $\mathsf{LogOfTrans}(t^1), \cdots, \mathsf{LogOfTrans}(t^h)$ is preserved in l. Since the only records of the form $(\_, \mathtt{write}(x, \_)$ which appear in l belong to one of the logs $\mathsf{LogOfTrans}(t^1), \cdots, \mathsf{LogOfTrans}(t^h)$, then

$$\mathsf{val}(x, l) = \mathsf{val}(x, \mathsf{LogOfTrans}(t^1) \cdot \cdots \cdot \mathsf{LogOfTrans}(t^h)) = n$$

– $(L, M) \xrightarrow{e} (L', M')$ because of Rule `(Commit)`. We need to show that $\mathsf{canCommit}(r, L)$ is satisfied. For any replica r, let $L(r) = (l_r, \varepsilon)$. We have to prove that, whenever l' contains a record of the form $(t_i, \mathtt{write}(x, \_))$, and l_r also contains a record of the form $(t_j, \mathtt{write}(x, \_))$, then l contains the record $(t_j, \mathtt{write}(x, \_))$. If l_r contains the record $(t_j, \mathtt{write}(x, \_))$, then such a record belongs to $\mathsf{LogOfTrans}(t_j)$; also, $t_j < t_i$.
Since $\mathcal{A} \in \mathsf{AbsPSI}$ (hence $\mathcal{A}$ satisfies (Wconflict)), then it has to be the case that $t_j \lhd_{\mathsf{hb}}^* t_i$. Therefore, it has to be the case that either $e_{\mathtt{receive}(t_j, t_i)} \in E_{\mathtt{receive}}$, or $e_{\mathtt{receive}(t_j, t_h)} \in E_{\mathtt{receive}}$ for some t_h such that $\mathsf{ReplOfTrans}(t_h) = \mathsf{ReplOfTrans}(t_i)$, and $t_j \lhd_{\mathsf{hb}}^* t_i$.
In both cases, we obtain that $\mathsf{LogOfTrans}(t_j)$ has been received at $\mathsf{ReplOfTrans}(t_i)$ before transaction t_i started, hence the record $(t_j, \mathtt{write}(x, \_))$ belongs to l.

Finally, we need to verify that the execution γ we have constructed satisfies the Transport Layer and Database Requirements (Figure 12). Property (OneStart) is satisfied, since we defined $\mathsf{TIdOf}(\cdot)$ to be one-to-one. Property (EndTr) is also satisfied, since for any $[e]_\sim in E_{\mathcal{A}/\sim}$, the transaction $\mathsf{TIdOf}([e]_\sim)$ eventually commits, via the event $e_{\mathtt{commit}(\mathsf{TIdOf}([e]_\sim)}$.

Property (NoSameRcv) is satisfied, because any receive event of the form $e_{\mathtt{receive}(t_1, t_2)}$ requires that $\mathsf{ReplOfTrans}(t_1) \neq \mathsf{ReplOfTrans}(t_2)$.

The definition of $E_{\texttt{receive}}$ also ensures that (OneRcv) is satisfied, since we require that if $e_{\texttt{receive}(t_1,t_2)} \in E_{\texttt{receive}}$, then there exists no t' such that $\mathsf{ReplOfTrans}(t') = \mathsf{ReplOfTrans}(t_2)$, and $t_1 \lhd^*_{\mathsf{hb}} t'$. In particular, this implies that there exists no t' such that $\mathsf{ReplOfTrans}(t') = \mathsf{ReplOfTrans}(t_2)$, and $e_{\texttt{receive}(t_1,t')} \in E_{\texttt{receive}}$.

Finally, Property (CDel) is also satisfied, because the order $<$ over transaction identifiers is preserved by $\lhd^*_{\mathsf{hb}}$, and messages which can be delivered to a given replica before some transaction starts, are delivered in the order established by $<$.

E Proof of Theorem 2

To prove that a particular history is allowed by PSI it is sufficient to find a happens-before relation such that extending the history with it yields an execution that satisfies the consistency axioms in Figure 4. The happens-before can be intuitively thought of as a *witness* that the history can be processed by the database according to the PSI rules. However, a given history may have multiple such witnesses. For example, the valid execution in Figure 13 will stay valid if we remove the two edges $e \xrightarrow{\mathsf{hb}_{\mathcal{A}}} f$ and $e \xrightarrow{\mathsf{hb}_{\mathcal{A}}} g$. For reasons that will be apparent later, to prove Theorem 2, we need a mechanism for finding a minimal happens-before that keeps a given execution valid. As it happens, we can define such a mechanisms using the relations introduced in Definition 7.

Definition 18. *The **minimisation** of* $\mathcal{A} = ((E, \mathsf{op}, \mathsf{co}, \sim), \mathsf{hb})$ *is*

$$\mu(\mathcal{A}) = ((E, \mathsf{op}, \mathsf{co}, \sim), \mathsf{co} \cup \langle(\mathsf{co} \cup \mathsf{rf} \cup \mathsf{vo}) \setminus \sim\rangle^+_\sim).$$

Theorem 5. *If* $\mathcal{A} \in \mathsf{AbsPSI}$*, then* $\mu(\mathcal{A}) \in \mathsf{AbsPSI}$*. Furthermore, if all values written in* $\mathcal{A}$ *are distinct:*

$$\forall e, f \in E_{\mathcal{A}}.\, \mathsf{op}(e) = \mathsf{op}(f) = \texttt{write}(\_,\_) \implies e = f,$$

then removing any edge from $\mathsf{hb}_{\mu(\mathcal{A})}$ *yields an invalid execution.*

For example, after removing the only edge from the execution $\mathcal{A}$ in Figure 13, we obtain $\mu(\mathcal{A})$. Thus, only chain order, reads-from and version order edges in happens-before actually matter for reproducing the outcome of a given history on PSI. Intuitively, this is a consequence of the axiom (Reads), which determines the value fetched by a read based solely on *writes* that happen before it, not *reads*.

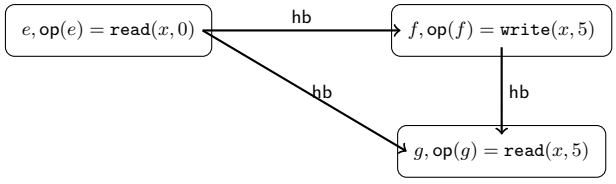

Fig. 13. A valid abstract execution $\mathcal{A}$ that stays valid after removing the edges $e \xrightarrow{\mathsf{hb}_{\mathcal{A}}} f$ and $e \xrightarrow{\mathsf{hb}_{\mathcal{A}}} g$, yielding $\mu(\mathcal{A})$.

We prove Theorem 5 by reformulating the axiomatic PSI specification of §3, so that as a witness for a history being allowed by PSI, we take the reads-from rf and version-order vo relations rather than happens-before. Then, instead of consistency axioms in Figure 4, we use the axioms in Figure 14, which define hb in terms of given rf and vo. Theorem 5 is a corollary of the following lemma, which establishes the equivalence of the two formulations.

Lemma 14. (i) *If* $\mathcal{A} = (\mathcal{H}, \_) \in \mathsf{AbsPSI}$*, then the properties in Figure 14 hold for* $\mathsf{rf} = \mathsf{rf}_{\mathcal{A}}$ *and* $\mathsf{vo} = \mathsf{vo}_{\mathcal{A}}$.

(ii) *If* $\mathcal{H}$ *and* $\mathsf{rf}, \mathsf{vo} \subseteq E_{\mathcal{H}} \times E_{\mathcal{H}}$ *validate the properties in Figure 14, then* $(\mathcal{H}, \mathsf{hb}) \in \mathsf{AbsPSI}$ *for* hb *defined in Figure 14.*

$$\mathsf{hb} := (\mathsf{co} \cup \langle(\mathsf{co} \cup \mathsf{rf} \cup \mathsf{vo}) \setminus \sim\rangle_{\sim})^{+}$$

$$\mathsf{vo} \text{ is a union of total orders on writes to each object} \tag{VOwf}$$

$$\forall e, f.\, e \xrightarrow{\mathsf{rf}} f \implies \exists x, n.\, \mathsf{op}(e) = \mathtt{write}(x, n) \wedge \mathsf{op}(f) = \mathtt{read}(x, n) \tag{RFwf1}$$

$$\forall e_1, e_2, f.\, e_1 \xrightarrow{\mathsf{rf}} f \wedge e_2 \xrightarrow{\mathsf{rf}} f \implies e_1 = e_2 \tag{RFunique}$$

$$\mathsf{hb} \cup \mathsf{rf} \cup \mathsf{vo} \text{ is acyclic} \tag{Acycl}$$

$$\neg\exists e, f, g.\ e \xrightarrow{\mathsf{vo}} f \xrightarrow{\mathsf{hb}} g,\ e \xrightarrow[\mathsf{rf}]{} g \tag{RFrecent}$$

$$\forall e, x, n.\, \mathsf{op}(e) = \mathtt{read}(x, n) \wedge (\neg\exists f.\, f \xrightarrow{\mathsf{rf}} e) \implies$$
$$n = 0 \wedge \neg\exists f.\, f \xrightarrow{\mathsf{hb}} e \wedge \mathsf{op}(f) = \mathtt{write}(x, \_) \tag{RFexists}$$

Fig. 14. Alternative PSI consistency axioms, constraining a history $\mathcal{H} = (E, \mathsf{op}, \mathsf{co}, \sim)$ and given reads-from and version-order relations $\mathsf{rf}, \mathsf{vo} \subseteq E \times E$.

Proof. We prove the two statements separately.

(i) Suppose that $\mathcal{A} = (\mathcal{H}, \mathsf{hb}_{\mathcal{A}}) \in \mathsf{AbsPSI}$ for some $\mathsf{hb}_{\mathcal{A}} \subseteq E_{\mathcal{A}} \times E_{\mathcal{A}}$; let hb be defined according to Figure 14, taking $\mathsf{rf} = \mathsf{rf}_{\mathcal{A}}$ and $\mathsf{vo} = \mathsf{vo}_{\mathcal{A}}$. We prove that each of the equations in Figure 14 is satisfied for $\mathsf{rf} := \mathsf{rf}_{\mathcal{A}}, \mathsf{vo} := \mathsf{vo}_{\mathcal{A}}$.

- (VOwf)**:** For any $x \in \mathsf{Obj}$, define $\mathsf{WR}^x_{\mathcal{A}} \triangleq \{e \in E_{\mathcal{A}} \mid \mathsf{op}(e) = \mathtt{write}(x, \_)\}$, and note that by definition $\mathsf{vo}_{\mathcal{A}} = \bigcup_{x \in \mathsf{Obj}} (\mathsf{hb}_{\mathcal{A}} \cap (\mathsf{WR}^x_{\mathcal{A}} \times \mathsf{WR}^x_{\mathcal{A}}))$. Also, by (Wconflict), for any $x \in \mathsf{Obj}$ the relation $\mathsf{hb}_{\mathcal{A}} \cap (\mathsf{WR}^x_{\mathcal{A}} \times \mathsf{WR}^x_{\mathcal{A}})$ is a total order over $\mathsf{WR}^x_{\mathcal{A}}$, which means that (VOwf) is satisfied for $\mathsf{vo} = \mathsf{vo}_{\mathcal{A}}$.
- (RFwf1)**:** If $e \xrightarrow{\mathsf{rf}_{\mathcal{A}}} f$, then $\mathsf{op}(f) = \mathtt{read}(x, n)$ for some object x. By Lemma 13 it holds that $e = \max_{\mathsf{hb}_{\mathcal{A}}}(\mathsf{PW}^x_{\mathcal{A}}(f))$, which in turn gives that $\mathsf{op}(e) = \mathtt{write}(x, m)$ for some m. Now it follows from Lemma 12 that $m = n$, so that (RFwf1) is satisfied for $\mathsf{rf} = \mathsf{rf}_{\mathcal{A}}$.
- (RFunique)**:** If $e_1 \xrightarrow{\mathsf{rf}_{\mathcal{A}}} f$, and $e_2 \xrightarrow{\mathsf{rf}_{\mathcal{A}}} f$ for some e_1, e_2, f, then by Lemma 13 we have that $f = \mathtt{read}(x, \_)$ for some $x \in \mathsf{Obj}$, and $e_1 = \max_{\mathsf{hb}_{\mathcal{A}}}(\mathsf{PW}^x_{\mathcal{A}}(f) = e_2$. Therefore, (RFunique) is satisfied for $\mathsf{rf} = \mathsf{rf}_{\mathcal{A}}$,

– (Acycl)**:** By definition, we have that $\mathsf{rf}_{\mathcal{A}} \subseteq \mathsf{hb}_{\mathcal{A}}, \mathsf{vo}_{\mathcal{A}} \subseteq \mathsf{hb}_{\mathcal{A}}$ and $\mathsf{co}_{\mathcal{A}} \subseteq \mathsf{hb}_{\mathcal{A}}$. By Lemma 1, the transitivity of $\mathsf{hb}_{\mathcal{A}}$, and because of (Atomic) and the fact that $(\cdot)^+$ is a closure operator, we have that

$$\langle(\mathsf{co}_{\mathcal{A}} \cup \mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})\backslash\sim\rangle_\sim \subseteq \langle\mathsf{hb}_{\mathcal{A}}\rangle_\sim \subseteq \langle\mathsf{hb}_{\mathcal{A}}\backslash\sim\rangle_\sim \subseteq \mathsf{hb}_{\mathcal{A}}$$

where we exploited the fact that $\mathsf{hb}_{\mathcal{A}}$ is transitive by hypothesis. Since both $\mathsf{hb} \subseteq \mathsf{hb}_{\mathcal{A}}, \mathsf{rf}_{\mathcal{A}} \subseteq \mathsf{hb}_{\mathcal{A}}$ and $\mathsf{vo}_{\mathcal{A}} \subseteq \mathsf{hb}_{\mathcal{A}}$, and since $\mathsf{hb}_{\mathcal{A}}$ is acyclic by hypothesis, then $\mathsf{hb} \cup \mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}}$ is also acyclic, as required by (Acycl).
By (Chains) we also know that $\mathsf{co}_{\mathcal{A}} \subseteq \mathsf{hb}_{\mathcal{A}}$; this inclusion, together with the inclusion above, gives $\mathsf{co} \cup \langle\mathsf{co} \cup \mathsf{rf} \cup \mathsf{vo}\backslash\sim\rangle_\sim \subseteq \mathsf{hb}_{\mathcal{A}}$, and by taking the transitive closure of both sides of this inclusion we get

$$(\mathsf{co} \cup \langle\mathsf{co} \cup \mathsf{rf} \cup \mathsf{vo}\backslash\sim\rangle_\sim)^+ \subseteq \mathsf{hb}_{\mathcal{A}}^+ \subseteq \mathsf{hb}$$

– (RFrecent)**:** Let $e \xrightarrow{\mathsf{rf}_{\mathcal{A}}} g$, $e \xrightarrow{\mathsf{vo}_{\mathcal{A}}} f \xrightarrow{\mathsf{hb}} g$ for some $e, f, g \in E_{\mathcal{A}}$. By Lemma 13 we have that $e = \max_{\mathsf{hb}_{\mathcal{A}}}(\mathsf{PW}^x_{\mathcal{A}} g$, which also gives that $\mathsf{op}(e) = \mathtt{write}(x, \_)$. Since $e \xrightarrow{\mathsf{vo}_{aexec}} f$, we also have that $\mathsf{op}(f) = \mathtt{write}(x, \_)$, and $e \xrightarrow{\mathsf{hb}_{\mathcal{A}}} f$. This means that $f \notin \mathsf{PW}^x_{\mathcal{A}}(g)$, which can be only in the case that $\neg(f \xrightarrow{\mathsf{hb}_{\mathcal{A}}} g)$. Since we have already proved that $\mathsf{hb} \subseteq \mathsf{hb}_{\mathcal{A}}$, we also have that $\neg(f \xrightarrow{\mathsf{hb}} g)$. Therefore, (RFrecent) is satisfied for $\mathsf{rf} = \mathsf{rf}_{\mathcal{A}}, \mathsf{vo} = \mathsf{vo}_{\mathcal{A}}$,
– (RFexists)**:** Suppose that $\mathsf{op}(e) = \mathtt{read}(x, n)$ and there exist no f such that $f \xrightarrow{\mathsf{rf}_{\mathcal{A}}} e$. By Lemma 13 it follows that $\max_{\mathsf{hb}_{\mathcal{A}}}(\mathsf{PW}^x_{\mathcal{A}}(e)) \uparrow$, and by Lemma 11 we infer that $\mathsf{PW}^x_{\mathcal{A}}(f) = \emptyset$. we have that Whenever $f \xrightarrow{\mathsf{hb}} e$ then $f \xrightarrow{\mathsf{hb}_{\mathcal{A}}} e$, and by definition of $\mathsf{PW}^x_{\mathcal{A}}(e)$ it is ensured that $\mathsf{op}(f) \neq \mathtt{write}(x, \_)$. Further, by applying Lemma 12 we obtain that that $n = 0$. Therefore, (RFexists) is satisfied for $\mathsf{rf} = \mathsf{rf}_{\mathcal{A}}$.

(ii) Let $\mathcal{H}$ be a history, and suppose that rf, vo are two relations over $E_{\mathcal{H}}$ which satisfy the axioms of Figure 14. We prove that each of the axiom of Figure 4 is satisfied by the abstract execution $\mathcal{A} = (\mathcal{H}, \mathsf{hb})$, where hb is defined according to Figure 14. Note also hb is acyclic because of (Acycl); it is also transitive by definition.
– (Chains)**:** this axiom is trivially satisfied, by definition of hb,
– (Wconflict)**:** let $e, f \in E_{\mathcal{H}}$ be two events such that $\mathsf{op}(e) = \mathtt{write}(x, \_), \mathsf{op}(f) = \mathtt{write}(x, \_)$ for some $x \in \mathsf{Obj}$. Assume that $e \neq f$. We can safely assume that $e \not\approx f$, for otherwise we would have that either $e \xrightarrow{\mathsf{co}_{\mathcal{H}}} f$ or $f \xrightarrow{\mathsf{co}_{\mathcal{H}}}$, from which either $e \xrightarrow{\mathsf{hb}} f$ or $f \xrightarrow{\mathsf{hb}} e$. Because of (VOwf), either $e \xrightarrow{\mathsf{vo}} f$ or $f \xrightarrow{\mathsf{vo}} e$. Without loss of generality, assume it is the first case. We have that $e \xrightarrow{\mathsf{vo}\backslash\approx} f$, which can be weakened to $e \xrightarrow{\mathsf{vo}\backslash\sim} f$; now it is not difficult to see that $e \xrightarrow{\mathsf{hb}} f$.
– (Atomic)**:** It suffices to show that

$$\langle(\mathsf{co}_{\mathcal{H}} \cup \langle(\mathsf{co}_{\mathcal{H}} \cup \mathsf{rf} \cup \mathsf{vo})\backslash\sim\rangle_\sim)\backslash\sim\rangle_\sim \subseteq \mathsf{co}_{\mathcal{H}} \cup \langle(\mathsf{co}_{\mathcal{H}} \cup \mathsf{rf} \cup \mathsf{vo})\backslash\sim\rangle_\sim$$

then the result follows from Corollary 3. By simple algebraic manipulation, and by Lemma 8 we can rewrite the LHS of the inclusion above as

$$\langle(\mathsf{co}_{\mathcal{H}} \setminus \sim) \cup \langle(\mathsf{co}_{\mathcal{H}} \cup \mathsf{rf} \cup \mathsf{vo})\rangle_\sim\backslash\sim\rangle_\sim$$

since $\mathsf{co}_{\mathcal{H}} \subseteq \sim$, we can further simplify the term above, leading to

$$\langle\langle \mathsf{co}_{\mathcal{H}} \cup \mathsf{rf} \cup \mathsf{vo}\rangle_{\sim})\backslash\sim\rangle_{\sim}$$

We can now apply Lemma 8, followed by Lemma 1, and rewrite the expression above as

$$\langle \mathsf{co}_{\mathcal{H}} \cup \mathsf{rf} \cup \mathsf{vo}\rangle_{\sim}\backslash\sim$$

Thus we have proved that

$$\langle(\mathsf{co}_{\mathcal{H}} \cup \langle(\mathsf{co}_{\mathcal{H}} \cup \mathsf{rf} \cup \mathsf{vo})\backslash\sim\rangle_{\sim})\backslash\sim\rangle_{\sim} = \langle\langle \mathsf{co}_{\mathcal{H}} \cup \mathsf{rf} \cup \mathsf{vo}\rangle_{\sim}\backslash\sim\rangle_{\sim} \subseteq \mathsf{co}_{\mathcal{H}} \cup \langle(\mathsf{co}_{\mathcal{H}} \cup \mathsf{rf} \cup \mathsf{vo})\backslash\sim\rangle_{\sim}$$

– (Reads)**:** We actually prove that $\mathcal{A}$ satisfies (RFmax). Since we have already proved that $\mathcal{A}$ satisfies (Wconflict), the result follows from Lemma 12. Let then $e \in E_{\mathcal{H}}$ be an event such that $\mathsf{op}(e) = \mathtt{read}(x, n)$. By Lemma 11 either $\mathsf{PW}^x_{\mathcal{A}}(e) = \emptyset$, or the event $f := \max_{\mathsf{hb}}(\mathsf{PW}^x_{\mathcal{A}}(e)$ exists. Suppose that $\mathsf{PW}^x_{\mathcal{A}}(e) = \emptyset$;we have to show that $n = 0$. By definition there exists no event f such that $f \xrightarrow{\mathsf{hb}_{\mathcal{A}}} e$ and $\mathsf{op}(f) = \mathtt{write}(x, \_)$. In particular, there exists no event f such that $f \xrightarrow{\mathsf{rf}} e$. By (RFexists) we have that $n = 0$, as we wanted to prove. Let now $f = \max_{\mathsf{hb}}(\mathsf{PW}^x_{\mathcal{A}}(e)$ be defined. Since $f \in \mathsf{PW}^x_{\mathcal{A}}(e)$, $\mathsf{op}(f) = \mathtt{write}(x, m)$ for some m. We need to show that $m = n$. In practice, it suffices to show that $f \xrightarrow{\mathsf{rf}} e$, and the result follows because of (RFwf1). First note that there exists an event g such that $g \xrightarrow{\mathsf{rf}} e$, because otherwise by (RFexists) we would have that $\mathsf{PW}^x_{\mathcal{A}}(e) = \emptyset$, contradicting the hypothesis. By definition of $\mathsf{PW}^x_{\mathcal{A}}(e)$, it has to be $g \in \mathsf{PW}^x_{\mathcal{A}}(\mathcal{A})$, hence $g \xrightarrow{\mathsf{hb}^*} f$. However, it cannot be $g \xrightarrow{\mathsf{hb}} f$; in fact, this would lead to $g \xrightarrow{\mathsf{vo}} f \xrightarrow{\mathsf{hb}} e$, and $g \xrightarrow{\mathsf{rf}} e$, contradicting axiom (RFrecent). Thus, $f = g$, which means that $f \xrightarrow{\mathsf{rf}} e$.

Thus, we could have defined abstract executions in Definition 2 as $(\mathcal{H}, \mathsf{rf}, \mathsf{vo})$ instead of $(\mathcal{H}, \mathsf{hb})$, and used the axioms in Figure 14 instead of Figure 4. Such a specification would be closer in style to the ones of weak consistency levels in SQL databases [2] and shared weak memory models [4]. We found that working with the specification based on happens-before was more convenient for proving Theorems 1 and the core of 2, whereas the specification based on reads-from and version order was more appropriate for the proof of Theorem 5.

We construct the splicing of a given execution $\mathcal{A}$ required by Definition 6 as follows.

Definition 19. *The **splicing** of $\mathcal{A} = ((E, \mathsf{op}, \mathsf{co}, \sim), \mathsf{hb})$ is*

$$\mathsf{splice}(\mathcal{A}) = ((E, \mathsf{op}, \mathsf{co}, \approx_{\mathcal{A}}), \mathsf{co} \cup \langle(\mathsf{rf} \cup \mathsf{vo}) \setminus \approx\rangle^{+}_{\approx_{\mathcal{A}}}),$$

where $\approx_{\mathcal{A}} = \mathsf{co} \cup \mathsf{co}^{-1} \cup \mathsf{Id}_E$, according to Definition 6.

Intuitively, to construct the happens-before relation in $\mathsf{splice}(\mathcal{A})$, we first minimise the happens-before in $\mathcal{A}$ according to Definition 18. Then, since the new same-transaction relation $\approx_{\mathcal{A}}$ is larger than $\sim_{\mathcal{A}}$, to satisfy the axiom Atomic and to keep the happens-before transitive, we factor it over $\approx_{\mathcal{A}}$ and take the transitive closure.

Theorem 2 then follows from

Theorem 6. *Given* $\mathcal{A} \in \mathsf{AbsPSI}$*, if the graph* $\mathsf{DCG}(\mathcal{A})$ *does not have critical cycles, then* $\mathsf{splice}(\mathcal{A}) \in \mathsf{AbsPSI}$.

Including happens-before minimisation into the construction of $\mathsf{splice}(\mathcal{A})$ is crucial for the above theorem to hold. Our proof of the theorem does not apply if we skip minimisation and just factor $\mathsf{hb}_{\mathcal{A}}$ over $\approx_{\mathcal{A}}$ and transitively close it; we conjecture that doing so leads to invalid executions even when $\mathsf{DCG}(\mathcal{A})$ does not have critical cycles.

First, we give a representation of the labels that can be encountered in a dynamic chopping graph, in terms of relations between transactions.

Definition 20 (Relations over Transactions). *Let* $\mathcal{A}$ *be an abstract execution; we define the following relations over* $(E_{\mathcal{A}})_{/\sim} \times (E_{\mathcal{A}})_{/\sim}$*:*

- $\mathbb{S}_{\mathcal{A}} = \mathsf{co}_{\mathcal{A}/\sim} \setminus \mathsf{Id}$,
- $\mathbb{P}_{\mathcal{A}} = \mathbb{S}_{\mathcal{A}}^{-1}$,
- $\mathbb{A}_{\mathcal{A}} = \mathsf{ad}_{\mathcal{A}/\sim} \setminus \approx_{/\sim}$,
- $\mathbb{D}_{\mathcal{A}} = (\mathsf{co}_{\mathcal{A}} \cup \mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})_{/\sim} \setminus \approx_{/\sim}$,
- $\mathbb{E}_{\mathcal{A}} = \mathbb{S}_{\mathcal{A}} \cup \mathbb{P}_{\mathcal{A}} \cup \mathbb{A}_{\mathcal{A}} \cup \mathbb{D}_{\mathcal{A}}$

Note that, if we also let $\mathbb{C}_{\mathcal{A}} = \mathbb{D}_{\mathcal{A}} \cup \mathbb{A}_{\mathcal{A}}$, then according to Definition 20 and Definition 8 a critical cycle in $\mathsf{DCG}(\mathcal{A})$ corresponds to a cycle in $\mathbb{E}_{\mu(\mathcal{A})}$ which contains a subpath labelled as $\mathbb{C}_{\mu(\mathcal{A})}\mathbb{P}_{\mu(\mathcal{A})}\mathbb{C}_{\mu(\mathcal{A})}$, and at most one $\mathbb{A}_{\mu(\mathcal{A})}$ edge. Theorem Theorem 2 can be restated as follows.

Theorem 7. *Let* $\mathcal{A} \in \mathsf{AbsPSI}$ *be an abstract execution. If* $\mathbb{E}_{\mu(\mathcal{A})}$ *does not contain any critical cycle, then* $\mathsf{splice}\mathcal{A} \in \mathsf{AbsPSI}$.

The proof of Theorem 7 is a consequence of lemmas 15, 16, 17, 20, and Corollary 6, which prove that the individual properties of any execution in $\mathcal{A} \in \mathsf{AbsPSI}$ by the operator $\mathsf{splice}(\cdot)$, under the assumption that $\mathbb{E}_{\mu(\mathcal{A})}$ does not contain a critical cycle.

Lemma 15. *Let* $\mathcal{A}$ *be an abstract execution. Let* $\mathsf{hb}' = \mathsf{hb}_{\mathsf{splice}(\mathcal{A})}$*. Then* $\langle(\mathsf{hb}')\backslash\approx\rangle_{\approx} \subseteq \mathsf{hb}'$.

Proof. By Lemma 2 it suffices to show that $\langle \mathsf{hb}'\backslash\approx\rangle_{\approx} \subseteq \mathsf{hb}'\backslash\approx$. By definition, $\mathsf{hb}' = \mathsf{co}_{\mathcal{A}} \cup \langle(\mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})\backslash\approx\rangle_{\approx}^{+}$. Therefore, we have to show that

$$\langle(\mathsf{co}_{\mathcal{A}} \cup \langle(\mathsf{rf} \cup \mathsf{vo})\backslash\approx\rangle_{\approx}^{+})\backslash\approx\rangle_{\approx} \subseteq (\mathsf{co}_{\mathcal{A}} \cup \langle(\mathsf{rf} \cup \mathsf{vo})\backslash\approx\rangle_{\approx}^{+})\backslash\approx$$

Since $\mathsf{co}_{\mathcal{A}} \subseteq\approx$, the inclusion above reduces to

$$\langle\langle(\mathsf{rf} \cup \mathsf{vo})\backslash\approx\rangle_{\approx}^{+}\backslash\approx\rangle_{\approx} \subseteq \langle(\mathsf{rf} \cup \mathsf{vo})\backslash\approx\rangle_{\approx}^{+}\backslash\approx$$

By Lemma Lemma 8 we have that

$$\langle\langle(\mathsf{rf} \cup \mathsf{vo})\backslash\approx\rangle_{\approx}^{+}\backslash\approx\rangle_{\approx} = \langle\langle(\mathsf{rf} \cup \mathsf{vo})\backslash\approx\rangle_{\approx}^{+}\rangle_{\approx}\backslash\approx$$

We can apply now Corollary 4 to the right hand side of the equation above, obtaining

$$\langle\langle(\mathsf{rf} \cup \mathsf{vo})\backslash\approx\rangle_{\approx}^{+}\rangle_{\approx}\backslash\approx = \langle(\mathsf{rf} \cup \mathsf{vo})\backslash\approx\rangle_{\approx}^{+}\backslash\approx$$

from which the inclusion we wanted to prove follows trivially.

Lemma 16. *Let $\mathcal{A}$ be an abstract execution, and let $\mathsf{hb}' = \mathsf{hb}_{\mathsf{splice}(\mathcal{A})}$. Then $(\mathsf{hb}')^+ \subseteq \mathsf{hb}'$.*

Proof. Since $\mathsf{co} \subseteq \approx$, and $\mathsf{hb}' = \mathsf{co} \cup \langle(\mathsf{rf} \cup \mathsf{vo}) \setminus \approx\rangle^+_{\approx}$, the result follows from Corollary 5.

Lemma 17. *Let $\mathcal{A} \in \mathsf{AbsPSI}$ be an abstract execution, and let $\mathcal{A}' = \mathsf{splice}(\mathcal{A})$. Then $hb_{\mathcal{A}'}$ satisfies Axiom* (Wconflict)*.*

Proof. Let $e, f \in E_{\mathcal{A}}$ be two events such that $\mathsf{op}(e) = \mathtt{write}(x, _), \mathsf{op}(f) = \mathtt{write}(x, _)$for some $x \in \mathsf{Obj}$, and such that $e \neq f$. We need to show that either $e \xrightarrow{\mathsf{hb}_{\mathcal{A}'}} f$, or $f \xrightarrow{\mathsf{hb}_{\mathcal{A}'}} e$.

By Axiom (Wconflict) (applied to the relation $\mathsf{hb}_{\mathcal{A}}$, we know that either $e \xrightarrow{\mathsf{hb}_{\mathcal{A}}} f$, or $f \xrightarrow{\mathsf{hb}_{\mathcal{A}}} e$. Without loss of generality, we can assume that it is the case that $e \xrightarrow{\mathsf{hb}_{\mathcal{A}}} f$. By definition, we have that $e \xrightarrow{\mathsf{vo}_{\mathcal{A}}} f$. There are two possible cases:

- $e \approx f$. In this case, note that it has to be the case that $e \xrightarrow{f}$. In fact, from $e \approx f$ and $e \neq f$ it follows that either $e \xrightarrow{so_{\mathcal{A}}} f$ of $f \xrightarrow{\mathsf{co}_{\mathcal{A}}} e$. But this last case is not possible, for Axiom (Chains) would imply that $f \xrightarrow{\mathsf{hb}_{\mathcal{A}}} e$; together with $e \xrightarrow{\mathsf{hb}_{\mathcal{A}}} f$, which we know to be true by hypothesis, we obtain a cycle in $\mathsf{hb}_{\mathcal{A}}$ contradicting the assumption that $\mathcal{A}$ is an abstract execution. Therefore $e \xrightarrow{\mathsf{co}_{\mathcal{A}}} f$, and by construction $e \xrightarrow{hb_{\mathcal{A}'}} f$, as we wanted to prove,
- $e \not\approx f$. By definition, we know that $e \xrightarrow{\mathsf{vo}} f$. It is not difficult now to infer that $e \approx e\ (\mathsf{rf} \cup \mathsf{vo})\setminus\approx f \approx f$, from which it follows that $e\ (\mathsf{co}_{\mathcal{A}} \cup \langle(\mathsf{rf} \cup \mathsf{vo})\setminus\approx\rangle_{\approx})^+\ f$. But this is exactly $e \xrightarrow{\mathsf{hb}_{\mathcal{A}'}} f$, as we wanted to prove.

Lemma 18. *Let $\mathcal{A} \in \mathsf{AbsPSI}$ be an abstract execution, and let $\mathcal{A}' = \mathsf{splice}(\mathcal{A})$. If $\mathsf{hb}_{\mathcal{A}'}$ has a cycle, then $\mathbb{E}_{\mathcal{A}}$ has a simple cycle which contains a subpath of the form $\mathbb{D}_{\mathcal{A}}\mathbb{P}_{\mathcal{A}}\mathbb{D}_{\mathcal{A}}$ and no edge $\mathbb{A}_{\mathcal{A}}$.*

Proof. First, note that since we are assuming that $\mathcal{A} \in \mathsf{AbsPSI}$, then by Theorem 5 $\mu(\mathcal{A}) \in \mathsf{AbsPSI}$. By Lemma 16, we know that $\mathsf{hb}_{\mathcal{A}'}$ is a transitive relation, so that by Lemma 5 the existence of a cycle in $\mathsf{hb}_{\mathcal{A}'}$ corresponds to the existence of an event e such that $e \xrightarrow{\mathsf{hb}_{\mathcal{A}'}} e$. Since $\mathsf{hb}_{\mathcal{A}'} = \mathsf{co}_{\mathcal{A}} \cup \langle(\mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})\setminus\approx\rangle^+_{\approx}$, and since $\mathsf{co}_{\mathcal{A}} \subseteq \mathsf{hb}_{\mathcal{A}}$, which is acyclic and transitive (hence irreflexive) by hypothesis, then it cannot be $e \xrightarrow{\mathsf{co}_{\mathcal{A}}} e$. Therefore, it has to be the case that $e\ \langle(\mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})\setminus\approx\rangle^+_{\approx}\ e$. Since, $\mathsf{co}_{\mathcal{A}}\subseteq\approx$, then $\mathsf{co}_{\mathcal{A}}\setminus\approx = \emptyset$, hence we also have that $e\ \langle(\mathsf{co}_{\mathcal{A}} \cup \mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})\setminus\approx\rangle^+_{\approx}\ e$ By Lemma 9 it follows that

$$[e]_{\sim}\ \langle(\mathsf{co}_{\mathcal{A}} \cup \mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})_{/\sim}\setminus\approx_{/\sim}\rangle^+_{\approx_{/\sim}}\ [e]_{\sim}$$

which can be rewritten, using the notation introduced in Definition 20, as

$$[e]_{\sim}\ \langle\mathbb{D}_{\mathcal{A}}\rangle^+_{\approx_{/\sim}}\ [e]_{\sim}$$

By Lemma 5 this amounts to say that there exists a cycle in the relation $\langle\mathbb{D}_{\mathcal{A}'}\rangle_{\approx_{/\sim}} = \approx_{/\sim}; \mathbb{D}_{\mathcal{A}'}; \approx_{/\sim}$. It is well known from graph theory that a cycle in $\langle\mathbb{D}_{\mathcal{A}}\rangle_{\approx_{/\sim}}$ can be

converted into a simple cycle. Furthermore, since the relation $\mathbb{D}_{\mathcal{A}'}$ never relates two elements $[f]_\sim, [f']_\sim$ such that $[f]_\sim \ (\approx_{/\sim}) \ [f']_\sim$, then such a simple cycle contains at least an occurrence of the relation $\mathbb{D}$. That is, there exist an index $n > 0$, and $[e_1]_\sim, [e'_1]_\sim, \cdots, [e'n]_\sim, [e']_n\sim$ such that $[e_1]_\sim = [e'_n]_\sim = [e]_\sim$, and

$$[e_1]_\sim \ (\approx_{/\sim}) \ [e'_1]_\sim \ \mathbb{D}_{\mathcal{A}} \ e_2 \ (\approx_{/\sim}) \cdots \mathbb{D}_{\mathcal{A}} \ [e_n]_\sim \ (\approx_{/\sim}) \ [e'_n]_\sim \tag{2}$$

Note that we have the equivalence $(\approx_{/\sim}) = \mathbb{S}_{\mathcal{A}'} \cup \mathbb{P}_{\mathcal{A}'} \cup \mathsf{Id}$. To see why this is true, note that

$$\begin{gathered}\mathbb{S}_{\mathcal{A}} \cup \mathbb{P}_{\mathcal{A}} \cup \mathsf{Id} = (\mathsf{co}_{\mathcal{A}/\sim} \backslash \mathsf{Id}) \cup ((\mathsf{co}_{\mathcal{A}/\sim})^{-1} \backslash \mathsf{Id}) \cup \mathsf{Id} = \\ \mathsf{co}_{\mathcal{A}/\sim} \cup (\mathsf{co}_{\mathcal{A}/\sim})^{-1} \cup \mathsf{Id} = \mathsf{co}_{\mathcal{A}/\sim} \cup (\mathsf{co}_{\mathcal{A}}^{-1})_{/\sim} \cup \mathsf{Id} = \\ (\mathsf{co}_{\mathcal{A}} \cup \mathsf{co}_{\mathcal{A}}^{-1} \cup \mathsf{Id})_{/\sim} = \approx_{/\sim}\end{gathered}$$

Thus, each of the $(\approx_{/\sim})$-edges in the Cycle (2) is either a $\mathbb{S}_{\mathcal{A}'}$-edge, a $\mathbb{P}_{\mathcal{A}'}$-edge, or a Id-edge.

Note that if all $(\approx_{/\sim})$-edge in the Cycle (2) were either $(\mathbb{S}_{\mathcal{A}} \cup \mathsf{Id})$-edges, then there would be a cycle in $\mathsf{hb}_{\mathcal{A}}$, contradicting the hypothesis that $\mathcal{A} \in \mathsf{AbsPSI}$. To see why this is true, note first that $[f]_\sim \ \mathbb{S}_{\mathcal{A}} \ [f']_\sim$ if and only if $f \ (\mathsf{co}_{\mathcal{A}} \backslash \sim \ f'$. This can be proved easily by showing that the relation $\mathsf{co}_{\mathcal{A}} \backslash \sim$ is atomic with respect to $\sim$, then employing Lemma 3. Note also that $[f]_\sim \ \mathsf{Id} \ [f']_\sim$ if and only if $f \sim f'$. Now we have that $\mathsf{co}_{\mathcal{A}} \backslash \sim \cup \sim = \mathsf{co} \cup \sim$, so that $[f] \ (\mathbb{S}_{\mathcal{A}} \cup \mathsf{Id}) \ [f']_\sim$ if and only if $f \ (\mathsf{co}_{\mathcal{A}} \cup \sim) \ f'$. Finally, note that if $[f]_\sim \ (\mathbb{D}_{\mathcal{A}}) \ [f']_\sim$, then $f \ (\mathsf{co}_{\mathcal{A}} \cup \mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}}) \ f'$, which means that $f \xrightarrow{\mathsf{hb}_{\mathcal{A}}} f'$. Also, $f \not\approx f'$, and since $\sim \subseteq \approx$ we also have that $f \not\sim f'$. Therefore, $f \xrightarrow{\mathsf{hb}_{\mathcal{A}} \backslash \sim} f'$ The Cycle (2) can be rewritten as

$$\begin{gathered}e_1 \ (\ \mathsf{co}_{\mathcal{A}} \cup \sim) e'_1 \ (\mathsf{hb}_{\mathcal{A}} \backslash \sim) \ e_2 \ (\ \mathsf{co}_{\mathcal{A}} \cup \sim) \cdots \\ \cdots \ (\mathsf{hb}_{\mathcal{A}} \backslash \sim) \ e_n \ (\ \mathsf{co}_{\mathcal{A}} \cup \sim) e'_n\end{gathered} \tag{3}$$

For any $i = 1, \cdots, n$, we have two possible cases:

- $e_i \ \mathsf{co}_{\mathcal{A}} \ e'_i$, which means that $e_i \xrightarrow{\mathsf{hb}_{\mathcal{A}}} e'_i$ because of Axiom (Chains); Also, $e'_i \xrightarrow{\mathsf{hb}_{\mathcal{A}} \backslash \sim} e_{i+1}$, which can be weakened to $e'_i \xrightarrow{\mathsf{hb}_{\mathcal{A}}} e_{i+1}$; by the transitivity of $\mathsf{hb}_{\mathcal{A}}$, we have that $e_i \xrightarrow{\mathsf{hb}_{\mathcal{A}}} e_{i+1}$,
- $e_i \sim e'_i$, in which case the fact that $e'_i \xrightarrow{\mathsf{hb}_{\mathcal{A}} \backslash \sim} e_{i+1}$, and Axiom (Atomic), lead to $e_i \xrightarrow{\mathsf{hb}_{\mathcal{A}}} e_{i+1}$.

We can also prove that $e_{n-1} \xrightarrow{\mathsf{hb}}_{\mathcal{A}} e1$ in a way similar to the one above, using the fact that $e'_n \sim e_1$. We have proved that we can rewrite Cycle (3) as

$$e_1 \xrightarrow{\mathsf{hb}_{\mathcal{A}}} e_2 \xrightarrow{\mathsf{hb}_{\mathcal{A}}} \xrightarrow{\mathsf{hb}_{\mathcal{A}}} \cdots e_{n-1} \xrightarrow{\mathsf{hb}_{\mathcal{A}}} e_1 \tag{4}$$

that is, that the relation $\mathsf{hb}_{\mathcal{A}}$ has a cycle.

We have proved that Cycle (2) contains at least a $\mathbb{P}_{\mathcal{A}}$-edge. Also, it contains at least a $\mathbb{D}_{\mathcal{A}}$-edge by construction, In order to show that this cycle has a subpath of the form

$\mathbb{D}_{\mathcal{A}}\mathbb{P}_{\mathcal{A}}\mathbb{D}_{\mathcal{A}}$ it is sufficient to show that it contains at least a second $\mathbb{D}_{\mathcal{A}}$-edge; recall, in fact, that Cycle (2) alternates $\mathbb{D}_{\mathcal{A}}$-edges with $(\mathbb{S}_{\mathcal{A}} \cup \mathbb{P}_{\mathcal{A}} \cup \mathsf{Id})$ edges.

'To prove that Cycle (2) has at least two $\mathbb{D}_{\mathcal{A}}$ edges, suppose that Cycle (2) has only one $\mathbb{D}_{\mathcal{A}}$-edge. Then (2) can be rewritten as $[e_1]_\sim\ (\approx_{/\sim})\ [e'_1]_\sim\ \mathbb{D}_{\mathcal{A}}\ [e_2]_\sim\ (\approx_{/\sim})\ [e'_2]_\sim$, where $[e'_2]_\sim = [e_1]_\sim$. Recall that, for $i = 1, 2$, $[e_i]_\sim\ (\mathbb{S}_{\mathcal{A}} \cup \mathbb{P}_{\mathcal{A}} \cup \mathsf{Id})\ [e'_i]_\sim$ is equivalent to $[e_i]_\sim\ (\approx_{/\sim}\ [e'_i]_\sim$; also, $[e'_1]_\sim\ \mathbb{D}_{\mathcal{A}}\ [e_2]_\sim$ implies that $\neg([e_1]_\sim\ (\approx_{/\sim})\ [e'_1]_\sim)$. Since $\approx_{/\sim}$ is an equivalence (Lemma 9), it follows that $\neg([e_1]_\sim\ (\approx_{/\sim})\ [e_2]_\sim$, leading to a contradiction.

Corollary 6. *Let $\mathcal{A} \in \mathsf{AbsPSI}$ be an abstract execution, and let $\mathcal{A}' = \mathsf{splice}(\mathcal{A})$. If $\mathsf{hb}_{\mathcal{A}'}$ has a cycle, then $\mathbb{E}_{\mu(\mathcal{A})}$ has a simple cycle which contains a subpath of the form $\mathbb{D}_{\mu(\mathcal{A})}\mathbb{P}_{\mu(\mathcal{A})}\mathbb{D}_{\mu(\mathcal{A})}$ and no edge $\mathbb{A}_{\mu(\mathcal{A})}$.*

Proof. This is a trivial consequence of Lemma 18, and the fact that there is a one-to-one correspondence between $\mathbb{D}_{\mathcal{A}}$ and $\mathbb{D}_{\mu(\mathcal{A})}$ edges, and similarly between $\mathbb{P}_{\mathcal{A}}$ $(\mathbb{S}_{\mathcal{A}})$ and $\mathbb{P}_{\mu(\mathcal{A})}$ $(\mathbb{S}_{\mu(\mathcal{A})})$ edges.

Lemma 19. *For any abstract execution $\mathcal{A} \in \mathsf{AbsPSI}$, $\mathsf{hb}_{\mu(\mathcal{A})} \subseteq \mathsf{hb}_{\mathsf{splice}(\mathcal{A})}$.*

Proof. Recall that $\mathsf{hb}_{\mu(\mathcal{A})} = (\mathsf{co}_{\mathcal{A}} \cup \langle(\mathsf{co}_{\mathcal{A}} \cup \mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})\backslash\sim\rangle_\sim)^+$, and $\mathsf{hb}_{\mathsf{splice}(\mathcal{A})} = \mathsf{co}_{\mathcal{A}} \cup \langle(\mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})\backslash\approx\rangle^+_\approx$. By Corollary 5, and using the fact that $\mathsf{co} \subseteq\approx$, hence $\mathsf{co}\backslash \approx = \emptyset$, we can rewrite $\mathsf{hb}_{\mathsf{splice}(\mathcal{A})}$ as

$$\mathsf{hb}_{\mathsf{splice}(\mathcal{A})} = (\mathsf{co}_{\mathcal{A}} \cup \langle(\mathsf{co}_{\mathcal{A}} \cup \mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})\backslash\approx\rangle_\approx)^+$$

Then, in order to prove that $\mathsf{hb}_{\mu(\mathcal{A})} \subseteq \mathsf{hb}_{\mathsf{splice}(\mathcal{A})}$, it suffices to prove that

$$(\mathsf{co}_{\mathcal{A}} \cup \langle(\mathsf{co}_{\mathcal{A}} \cup \mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})\backslash\sim\rangle_\sim) \subseteq (\mathsf{co}_{\mathcal{A}} \cup \langle(\mathsf{co}_{\mathcal{A}} \cup \mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})\backslash\approx\rangle_\approx)$$

the result then follows from the fact that $(\cdot)^+$ is a closure operator.

Suppose then that $e\ \ (\mathsf{co}_{\mathcal{A}} \cup \langle(\mathsf{co}_{\mathcal{A}} \cup \mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})\backslash\sim\rangle_\sim)\ \ f$. There are two possibilities: **(i)** either $e \xrightarrow{\mathsf{co}} f$, from which it follows trivially that $e\ \ (\mathsf{co}_{\mathcal{A}} \cup \langle(\mathsf{co}_{\mathcal{A}} \cup \mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})\backslash\approx\rangle_\approx)\ \ f$, or **(ii)** there exist $e' \sim e, f' \sim f$ such that $e' \not\sim f'$ and $e' \xrightarrow{(\mathsf{co}_{\mathcal{A}} \cup \mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})} f'$. Note that we also have that $e \approx e', f \approx f'$; now we have two possible sub-cases:

- $e' \not\approx f'$, in which case $e \approx e' \xrightarrow{(\mathsf{co}_{\mathcal{A}} \cup \mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})} f' \approx e$, or equivalently $e\ \langle(\mathsf{co}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}} \cup \mathsf{rf}_{\mathcal{A}})\backslash\approx\rangle_\approx f$,
- $e' \approx f'$; in this case we also have that $e \approx f$. Theorem 5 ensures that $\mathsf{hb}_{\mu(\mathcal{A})} \in \mathsf{AbsPSI}$, hence $\mathsf{hb}_{\mu(\mathcal{A})}$ is acyclic. A consequence of this is that $\mathsf{hb}_{\mu(\mathcal{A})} \cap \approx\ = \mathsf{co}_{\mathcal{A}}$, from which it follows that $e\mathsf{co}_{\mathcal{A}} f$, hence $e(\mathsf{co}_{\mathcal{A}} \cup \langle(\mathsf{co}_{\mathcal{A}} \cup \mathsf{rf}_{\mathcal{A}} \cup \mathsf{vo}_{\mathcal{A}})\backslash\approx\rangle_\approx)$.

Lemma 20. *Let $\mathcal{A}$ be an abstract execution in AbsPSI, and suppose that $\mathsf{hb}_{\mathsf{splice}(\mathcal{A})}$ is acyclic. If $\mathsf{splice}(\mathcal{A})$ does not satisfy* (RFmax)*, then there exists two transactions $[e]_\sim$, $[f]_\sim$ such that $[e]_\sim \xrightarrow{\mathbb{A}_{\mu(\mathcal{A})}} [f]_\sim$ such that $[f]_\sim \xrightarrow{(\mathbb{D};\approx_{/\sim})^*;\mathbb{P}_{\mu(\mathcal{A})};(\mathbb{D};\approx_{/\sim})^*} [e]_\sim$,*

Proof. First note that $\mu(\mathcal{A}) \in \mathsf{AbsPSI}$, by Theorem 5. If $\mathsf{splice}(\mathcal{A})$ does not satisfy (RFmax), then there exist e, x, n such that $\mathsf{op}(e) = \mathtt{read}(x, n)$ and either **(i)** $\mathsf{PW}^x_{\mathsf{splice}(\mathcal{A})}(e) = \emptyset$ and $n \neq 0$, or **(ii)** let $f; = \max_{\mathsf{hb}_{\mathsf{splice}(\mathcal{A})}} \mathsf{PW}^x_{\mathsf{splice}(\mathcal{A})}(e)$, then $\mathsf{op}(f) = \mathtt{write}(x, m)$ for some $m \neq n$.

First, we note that case **(i)** is not possible. In fact, by Lemma 19 we know that $\mathsf{hb}_{\mu(\mathcal{A})} \subseteq \mathsf{hb}_{\mathsf{splice}(\mathcal{A})}$, from which it follows that $\mathsf{PW}^x_{\mu(\mathcal{A})}(e) \subseteq \mathsf{PW}^x_{\mathsf{splice}(\mathcal{A})}(e)$. Therefore, $\mathsf{PW}^x_{\mathsf{splice}(\mathcal{A})}(e) = \emptyset$ implies that $\mathsf{PW}^x_{\mu(\mathcal{A})}(e) = \emptyset$. Since $\mu(\mathcal{A}) \in \mathsf{AbsPSI}$, by Lemma 12 it satisfies (RFmax); we have that $\mathtt{read}(x, n) = \mathsf{op}(e) = \mathtt{read}(x, 0)$, hence $n = 0$.

Therefore, it has to be case **(ii)**. In this case we have that $e \xrightarrow{\mathsf{ad}_{\mu(\mathcal{A})}} f$. In fact, since $f = \max_{\mathsf{hb}_{\mathsf{splice}(\mathcal{A})}} \mathsf{PW}^x_{\mathsf{splice}(\mathcal{A})}(e)$, $\mathsf{hb}_{mu(\mathcal{A})} \subseteq \mathsf{hb}_{\mathsf{splice}(\mathcal{A})}$, and $\mathsf{PW}^x_{\mu(\mathcal{A})}(e) \subseteq \mathsf{PW}^x_{\mathsf{splice}(\mathcal{A})}(e)$, it follows that f is an upper-bound for $\mathsf{PW}^x_{\mu(\mathcal{A})}(e)$, with respect to $\mathsf{hb}_{\mu(\mathcal{A})}$; $\forall f' \in \mathsf{PW}^x_{\mu(\mathcal{A})}(e). f' \xrightarrow{\mathsf{hb}_{\mu(\mathcal{A})}} e$. We also have that $f \notin \mathsf{PW}^x_{\mu(\mathcal{A})}(e)$, because otherwise it would be $f = \max_{\mathsf{hb}_{\mu(\mathcal{A})}}(\mathsf{PW}^x_{\mu(\mathcal{A})}(e))$; since $\mathsf{op}(f) = \mathtt{read}(x, m)$, where $m \neq n$, this would contradict the hypothesis that $\mu(\mathcal{A})$ satisfies (RFmax). Since $f \notin \mathsf{PW}^x_{\mu(\mathcal{A})}(e)$, and $\mathsf{op}(f) = \mathtt{write}(x, m)$, by definition of $\mathsf{PW}^x_{\mu(\mathcal{A})}(e)$ it follows that $\neg(f \xrightarrow{\mathsf{hb}_{\mu(\mathcal{A})}} e$. At this point we have that $\mathsf{op}(f) = \mathtt{write}(x, m)$, $\mathsf{op}(e) = \mathtt{read}(x, n)$ and $\neg(f \xrightarrow{\mathsf{hb}_{\mu(\mathcal{A})}} e)$, which by definition is $e \xrightarrow{\mathsf{ad}_{\mu(\mathcal{A})}} f$.

Since $\mathsf{op}(f) = \mathtt{write}(x, m), \mathsf{op}(e) = \mathtt{write}(x, n)$, then $e \neq f$. Also, since $f \xrightarrow{\mathsf{hb}_{\mathsf{splice}(\mathcal{A})}} e$, it cannot be $e \xrightarrow{\mathsf{co}_{\mathsf{splice}(\mathcal{A})}} f$, for this would contradict the hypothesis that $\mathsf{hb}_{\mathsf{splice}(\mathcal{A})}$ is acyclic; since $\mathsf{co}_{\mathsf{splice}(\mathcal{A})} = \mathsf{co}_{\mathcal{A}} = \mathsf{co}_{\mu(\mathcal{A})}$, we have that $\neg(e \xrightarrow{\mathsf{co}_{\mu(\mathcal{A})}} f)$. Also, it cannot be $f \xrightarrow{\mathsf{co}_{\mu(\mathcal{A})}} e$, for this would imply that $f \xrightarrow{\mathsf{hb}_{\mu(\mathcal{A})}} e$, hence $f \in \mathsf{PW}^x_{\mu(\mathcal{A})}(e)$, which we already proved to be false. Thus, e, f are not related by Id, co, nor by co^{-1}; as a consequence $e \not\approx f$. We have proved that $e \xrightarrow{\mathsf{ad}_{\mu(\mathcal{A})}} f$, and $e \not\approx f$. That is, $[e]_\sim \xrightarrow{\mathbb{A}_{\mu(\mathcal{A})}} [f]_\sim$.

Since $f \xrightarrow{\mathsf{hb}_{\mathsf{splice}(\mathcal{A})}} e$, and $[f]_\approx \neq [e]_\approx$, we can find a sequence of events $g_0, g'_0, \cdots, g_k, g'_k$ such that $f = g_0 \approx g'_0 \xrightarrow{(\mathsf{hb}_{\mu(\mathcal{A})} \setminus \approx_{/\sim})} g_1 \approx g'_1 \xrightarrow{((\mathsf{hb}_{\mu(\mathcal{A})} \setminus \approx_{/\sim})} \cdots \xrightarrow{(\mathsf{hb}_{\mu(\mathcal{A})} \setminus \approx_{/\sim})} g_k \approx g'_k = e$. there exist two events f', e' such that $f' \approx f$, $e' \approx e$. This implies that there exists a path from $[f]_\sim$ to $[e]_\sim$ of the form $[f]_\sim \xrightarrow{(\mathbb{D}_{\mu(\mathcal{A})} \approx_{/\sim})^*} [e]_\sim$. It remains to show that, in the path $[f]_\sim \xrightarrow{(\mathbb{D}_{\mu(\mathcal{A})} \approx_{/\sim})^+} [e]_\sim$, at least one edge is labelled as $\mathbb{P}_{\mu(\mathcal{A})}$. We can proceed in the same way of Lemma Lemma 18, showing that if there were no $\mathbb{P}_{\mu(\mathcal{A})}$ edges in the path above, then it would follow that $[f]_\sim \xrightarrow{(\mathsf{hb}_{\mu(\mathcal{A})})_{/\sim}} [e]_\sim$; since we already know that $f \not\approx e$, hence $\neg([f]_\sim \approx_{/\sim} [e]_\sim)$, it has to be the case that $f \xrightarrow{(\mathsf{hb}_{\mu(\mathcal{A})} \setminus \approx)} e$, from which $f \xrightarrow{\mathsf{hb}_{\mu(\mathcal{A})}} e$ follows trivially. But this contradict the assumption that $e \xrightarrow{\mathsf{ad}_{\mu(\mathcal{A})}} f$; therefore, at least one edge in the path $[f]_\sim \xrightarrow{(\mathbb{D}_{\mu(\mathcal{A})} \approx_{/\sim})^+} [e]_\sim$ has to be labelled as $\mathbb{P}_{\mu(\mathcal{A})}$.