On the Complexity of Checking Mixed Isolation Levels for SQL Transactions

Ahmed Bouajjani[1], Constantin Enea[2], and Enrique Román-Calvo[1]

[1] Université Paris Cité, CNRS, IRIF
{abou, calvo}@irif.fr
[2] LIX, École Polytechnique, CNRS and
Institut Polytechnique de Paris
cenea@lix.polytechnique.fr

Abstract. Concurrent accesses to databases are typically grouped in transactions which define units of work that should be isolated from other concurrent computations and resilient to failures. Modern databases provide different levels of isolation for transactions that correspond to different trade-offs between consistency and throughput. Quite often, an application can use transactions with different isolation levels at the same time. In this work, we investigate the problem of testing isolation level implementations in databases, i.e., checking whether a given execution composed of multiple transactions adheres to the prescribed isolation level semantics. We particularly focus on transactions formed of SQL queries and the use of multiple isolation levels at the same time. We show that many restrictions of this problem are NP-complete and provide an algorithm which is exponential-time in the worst-case, polynomial-time in relevant cases, and practically efficient.

1 Introduction

Concurrent accesses to databases are typically grouped in transactions which define units of work that should be isolated from other concurrent computations and resilient to failures. Modern databases provide different levels of isolation for transactions with different trade-offs between consistency and throughput. The strongest isolation level, *Serializability* [21], provides the illusion that transactions are executed atomically one after another in a serial order. Serializability incurs a high cost in throughput. For performance, databases provide weaker isolation levels, e.g., *Snapshot Isolation* [6] or *Read Committed* [6].

The concurrency control protocols used in large-scale databases to implement isolation levels are difficult to build and test. For instance, the black-box testing framework Jepsen [19] found a remarkably large number of subtle problems in many production databases.

In this work, we focus on testing the isolation level implementations in databases, and more precisely, on the problem of checking whether a given execution adheres to the prescribed isolation level semantics. Inspired by scenarios that arise in commercial software [22], we consider a quite generic version of the problem where transactions are formed of SQL queries and *multiple* isolation levels are used at the same time, i.e., each transaction is assigned a possibly different isolation level (the survey in [22] found that 32% of the respondents use such “heterogeneous” configurations). Previous work [21,7] studied the complexity of the problem when transactions are formed of reads and writes on a *static* set of keys (variables), and all transactions have the *same* isolation level.

As a first contribution, we introduce a formal semantics for executions with SQL transactions and a range of isolation levels, including serializability, snapshot isolation, prefix consistency, and read committed. Dealing with SQL queries is more challenging than classic reads and writes of a *static* set of keys (as assumed in previous formalizations [11,7]). SQL insert and delete queries change the set of locations at runtime and the set of locations returned by an SQL query depends on their values (the values are restricted to satisfy `WHERE` clauses).

We define an abstract model for executions, called *history*, where every SQL query that inspects the database (has a `WHERE` clause) is associated with a set of SQL queries that wrote the inspected values. This relation is called a *write-read* relation (also known as read-from). This is similar to associating reads to writes in defining memory models. We consider two classes of histories depending on the “completeness” of the write-read relation. To define a formal semantics of isolation levels, we need a complete write-read relation in the sense that for instance, an SQL select is associated with a write *for every* possible row (identified by its primary key) in the database, even if that row is *not* returned by the select because it does not satisfy the `WHERE` clause. Not returning a row is an observable effect that needs to be justified by the semantics. Such *full* histories can not be constructed by interacting with the database in a black-box manner (a desirable condition in testing) when only the outputs returned by queries can be observed. Therefore, we introduce the class of *client* histories where the write-read concerns only rows that are *returned* by a query. The consistency of a client history is defined as the existence of an extension of the write-read to a full history which satisfies the semantics. The semantics on full histories combines axioms from previous work [7] in a way that is directed by SQL queries that inspect the database and the isolation level of the transaction they belong to. This axiomatic semantics is validated by showing that it is satisfied by a standard operational semantics inspired by real implementations.

We study the complexity of checking if a full or client history is consistent, it satisfies the prescribed isolation levels. This problem is more complex for client histories, which record less dependencies and need to be extended to full ones.

For full histories, we show that the complexity of consistency checking matches previous results in the reads and writes model when all transactions have the same isolation level [7]: polynomial time for the so-called saturable isolation levels, and NP-complete for stronger levels like Snapshot Isolation or Serializability. The former is a new result that generalizes the work of [7] and exposes the key ideas for achieving polynomial-time complexity, while the latter is a consequence of the previous results.

We show that consistency checking becomes NP-complete for client histories even for saturable isolation levels. It remains NP-complete regardless of the expressiveness of `WHERE` clauses (for this stronger result we define another class of histories called *partial-observation*). The problem is NP-complete even if we bound the number of sessions. In general, transactions are organized in *sessions* [23], an abstraction of the sequence of transactions performed during the execution of an application (the counterpart of threads in shared memory). This case is interesting because it is polynomial-time in the read/write model [7].

As a counterpart to these negative results, we introduce an algorithm for checking consistency of client histories which is exponential-time in the worst case, but polynomial time in relevant cases. Given a client history as input, this algorithm combines an enumeration of extensions towards a full history with a search for a total commit order that satisfies the required axioms. The commit order represents the order in which transactions are committed in the database and it is an essential artifact for defining isolation levels. For efficiency, the algorithm uses a non-trivial enumeration of extensions that are *not* necessarily full but contain enough information to validate consistency. The search for a commit order is a non-trivial generalization of an algorithm by Biswas et al. [7] which concerned only serializability. This generalization applies to all practical isolation levels and combinations thereof. We evaluate an implementation of this algorithm on histories generated by PostgreSQL with a number of applications from BenchBase [12], e.g., the TPC-C model of a store and a model of Twitter. This evaluation shows that the algorithm is quite efficient in practice and scales well to typical workloads used in testing databases.

To summarize, we provide the first results concerning the complexity of checking the correctness of mixed isolation level implementations for SQL transactions. We introduce a formal specification for such implementations, and a first tool that can be used in testing their correctness.

2 Histories

2.1 Transactions

We model the database as a set of rows from an unbounded domain Rows. Each row is associated to a unique (primary) key from a domain Keys, given by the function key : Rows → Keys. We consider client programs accessing the database from a number of parallel sessions, each session being a sequence of transactions defined by the following grammar:

$$\iota \in \mathsf{Iso} \quad a \in \mathsf{LVars} \quad \mathsf{R} \in 2^{\mathsf{Rows}} \quad \mathsf{p} \in \mathsf{Rows} \to \{0,1\} \quad \mathsf{U} \in \mathsf{Keys} \to \mathsf{Rows}$$

$$\begin{array}{rl}
\mathsf{Transaction} ::= & \texttt{begin}(\iota); \mathsf{Body}; \texttt{commit} \\
\mathsf{Body} ::= & \mathsf{Instr} \ \mid \ \mathsf{Instr}; \mathsf{Body} \\
\mathsf{Instr} ::= & \mathsf{InstrDB} \ \mid \ a := \mathsf{LExpr} \mid \ \texttt{if}(\mathsf{LCond})\{\mathsf{Instr}\} \\
\mathsf{InstrDB} ::= & a := \texttt{SELECT}(\mathsf{p}) \ \mid \ \texttt{INSERT}(\mathsf{R}) \ \mid \ \texttt{DELETE}(\mathsf{p}) \ \mid \ \texttt{UPDATE}(\mathsf{p},\ \mathsf{U}) \ \mid \texttt{abort}
\end{array}$$

Each transaction is delimited by `begin` and `commit` instructions. The `begin` instruction defines an isolation level ι for the current transaction. The set of isolation levels Iso we consider in this work will be defined later. The body contains standard SQL-like statements for accessing the database and standard assignments and conditionals for local computation. Local computation uses (transaction-)local variables from a set LVars. We use a, b, ... to denote local variables. Expressions and Boolean conditions over local variables are denoted with LExpr and LCond, respectively.

Concerning database accesses (sometimes called queries), we consider a simplified but representative subset of SQL: `SELECT`(p) returns the set of rows satisfying the predicate p and the result is stored in a local variable a. `INSERT`(R) inserts the set of rows R or updates them in case they already exist (this corresponds to `INSERT ON CONFLICT DO UPDATE` in PostgreSQL) , and `DELETE`(p) deletes all the rows that satisfy p. Then, `UPDATE`(p, U) updates the rows satisfying p with values given by the map U, i.e., every row r in the database that satisfies p is replaced with $\mathsf{U}(\mathsf{key}(\mathsf{r}))$, and `abort` aborts the current transaction. The predicate p corresponds to a `WHERE` clause in standard SQL.

2.2 Histories

We define a model of the interaction between a program and a database called *history* which abstracts away the local computation in the program and the internal behavior of the database. A history is a set of *events* representing the database accesses in the execution grouped by transaction, along with some relations between these events which explain the output of `SELECT` instructions.

An event is a tuple $\langle e, \mathit{type}\rangle$ where e is an *identifier* and *type* is one of `begin`, `commit`, `abort`, `SELECT`, `INSERT`, `DELETE` and `UPDATE`. $\mathcal{E}$ denotes the set of events. For an event e of type `SELECT`, `DELETE`, or `UPDATE`, we use `WHERE`(e) to denote the predicate p and for an `UPDATE` event e, we use `SET`(e) to denote the map U.

We call `read` events the `SELECT` events that read the database to return a set of rows, and the `DELETE` and `UPDATE` events that read the database checking satisfaction of some predicate p. Similarly, we call `write` events the `INSERT`, `DELETE` and `UPDATE` events that modify the database. We also say that an event is of type `end` if it is either a `commit` or an `abort` event.

A *transaction log* $(t, \iota_t, E, \mathsf{po}_t)$ is an identifier t, an *isolation level* identifier ι_t, and a finite set of events E along with a strict total order po_t on E, called *program order* (representing the order between instructions in the body of a transaction). The set E of events in a transaction log t is denoted by $\mathsf{events}(t)$. For simplicity, we may use the term *transaction* instead of transaction log.

Isolation levels differ in the values returned by read events which are not preceded by a write on the same variable in the same transaction. We denote by $\mathsf{reads}(t)$ the set of `read` events contained in t. Also, if t does *not* contain an `abort` event, the set of `write` events in t is denoted by $\mathsf{writes}(t)$. If t contains an `abort` event, then we define $\mathsf{writes}(t)$ to be empty. This is because the effect of aborted transactions (its set of writes) should not be visible to other transactions. The extension to sets of transaction logs is defined as usual.

To simplify the exposition we assume that for any given key $x \in \mathsf{Keys}$, a transaction does not modify (insert/delete/update) a row with key x more than once. Otherwise, under all isolation levels, only the last among multiple updates is observable in other transactions.

As expected, we assume that the minimal element of po_t is a `begin` event, if a `commit` or an `abort` event occurs, then it is maximal in po_t, and a log cannot contain both `commit` and `abort`. A transaction log without `commit` or `abort` is called *pending*. Otherwise, it is *complete*. A complete transaction log with a `commit` is *committed* and *aborted* otherwise.

A *history* contains a set of transaction logs (with distinct identifiers) ordered by a (partial) *session order* so that represents the order between transactions in the same session. It also includes a *write-read* relation wr which associates `write` events with `read` events. The `write` events associated to a `read` implicitly define the values observed (returned) by the `read` (read events do *not* include explicit values). Let T be a set of transaction logs. For every key $x \in \mathsf{Keys}$ we consider a write-read relation $\mathsf{wr}_x \subseteq \mathsf{writes}(T) \times \mathsf{reads}(T)$. The union of wr_x for every $x \in \mathsf{Keys}$ is denoted by wr. We extend the relations wr and wr_x to pairs of transactions by $(t_1, t_2) \in \mathsf{wr}$, resp., $(t_1, t_2) \in \mathsf{wr}_x$, iff there exist events w in t_1 and r in $t_2, t_2 \neq t_1$ s.t. $(w, r) \in \mathsf{wr}$, resp., $(w, r) \in \mathsf{wr}_x$. Analogously, we extend wr and wr_x to tuples formed of a transaction (containing a write) and a read event. We say that the transaction t_1 is *read* by the transaction t_2 when $(t_1, t_2) \in \mathsf{wr}$. The inverse of wr_x is defined as usual and denoted by wr_x^{-1}. We assume that wr_x^{-1} is a partial function and thus, use $\mathsf{wr}_x^{-1}(e)$ to denote the `write` event w such that $(w, e) \in \mathsf{wr}_x$. We also use $\mathsf{wr}_x^{-1}(e) \downarrow$ and $\mathsf{wr}_x^{-1}(e) \uparrow$ to say that there exists a `write` w such that $(w, e) \in \mathsf{wr}_x$ (resp. such `write` w does not exist).

To simplify the exposition, every history includes a distinguished transaction `init` preceding all the other transactions in so and inserting a row for every x. It represents the initial state and it is the only transaction that may insert as value $\dagger_x$ (indicating that initially, no row with key x is present).

Definition 1. *A* history $(T, \mathsf{so}, \mathsf{wr})$ *is a set of transaction logs T along with a strict partial* session order so, *and a* write-read *relation* $\mathsf{wr}_x \subseteq \mathsf{writes}(T) \times \mathsf{reads}(T)$ *for each* $x \in \mathsf{Keys}$ *s.t.*

- *the inverse of* wr_x *is a partial function,*
- $\mathsf{so} \cup \mathsf{wr}$ *is acyclic (here we use the extension of* wr *to pairs of transactions),*
- *if* $(w, r) \in \mathsf{wr}_x$*, then* $\mathtt{value}_{\mathsf{wr}}(w, x) \neq \bot$*, where*

$$\mathtt{value}_{\mathsf{wr}}(w, x) = \begin{cases} \mathsf{r} & \text{if } w = \mathtt{INSERT}(\mathsf{R}) \wedge \mathsf{r} \in \mathsf{R} \wedge \mathsf{key}(\mathsf{r}) = x \\ \dagger_x & \text{if } w = \mathtt{DELETE}(\mathsf{p}) \wedge \ \mathsf{wr}_x^{-1}(w) \downarrow \\ & \qquad \wedge\ \mathsf{p}(\mathtt{value}_{\mathsf{wr}}(\mathsf{wr}_x^{-1}(w), x)) = 1 \\ \mathsf{U}(x) & \text{if } w = \mathtt{UPDATE}(\mathsf{p},\ \mathsf{U}) \wedge \ \mathsf{wr}_x^{-1}(w) \downarrow \\ & \qquad \wedge\ \mathsf{p}(\mathtt{value}_{\mathsf{wr}}(\mathsf{wr}_x^{-1}(w), x)) = 1 \\ \bot & \text{otherwise} \end{cases}$$

The function wr_x^{-1} may be partial because some query may not read a key x, e.g., if the corresponding row does not satisfy the query predicate.

The function $\mathtt{value}_{\mathsf{wr}}(w, x)$ returns the row with key x written by the `write` event w. If w is an `INSERT`, it returns the inserted row with key x. If w is an

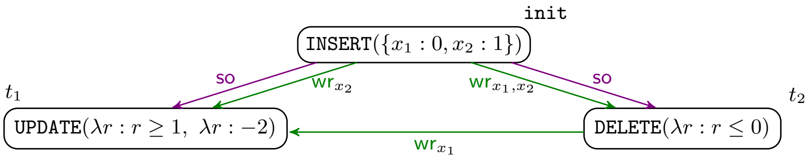

init
INSERT($\{x_1 : 0, x_2 : 1\}$)
t_1
so
wr_{x_2}
wr_{x_1,x_2}
so
t_2
UPDATE($\lambda r : r \geq 1,\ \lambda r : -2$)
DELETE($\lambda r : r \leq 0$)
wr_{x_1}

Fig. 1: An example of a history (isolation levels omitted for legibility). Arrows represent so and wr relations. Transaction $\texttt{init}$ defines the initial state: row 0 with key x_1 and row 1 with key x_2. Transaction t_2 reads x_1 and x_2 from $\texttt{init}$ and deletes row with key x_1 (the only row satisfying predicate $\lambda r : r \leq 0$ corresponds to key x_1). Transaction t_1 reads x_1 from t_2 and x_2 from $\texttt{init}$, and updates only row with key x_2 as this is the only row satisfying predicate $\lambda r : r \geq 1$.

$\texttt{UPDATE}(\mathsf{p},\ \mathsf{U})$ event, it returns the value of U on key x if w reads a value for key x that satisfies predicate p. If w is a $\texttt{DELETE}(p)$, it returns the special value $\dagger_x$ if w reads a value for key x that satisfies p. This special value indicates that the database does *not* contain a row with key x. In case no condition is satisfied, $\texttt{value}_{\mathsf{wr}}(w, x)$ returns an undefined value $\bot$. We assume that the special values $\dagger_x$ or $\bot$ do not satisfy any predicate. Note that the recursion in the definition of $\texttt{value}_{\mathsf{wr}}(w, x)$ terminates because wr is an acyclic relation.

Figure 1 shows an example of a history. For the $\texttt{UPDATE}$ event w in t_1, $\texttt{value}_{\mathsf{wr}}(w, x_1) = \bot$ because this event reads x_1 from the $\texttt{DELETE}$ event in t_2; while $\texttt{value}_{\mathsf{wr}}(w, x_2) = -2$ as it reads x_2 from the $\texttt{INSERT}$ event in $\texttt{init}$.

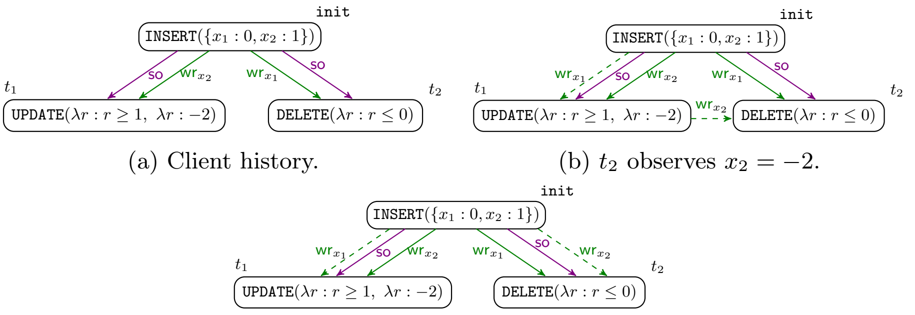
The set of transaction logs T in a history $h = (T, \mathsf{so}, \mathsf{wr})$ is denoted by $\mathsf{tr}(h)$ and $\mathsf{events}(h)$ is the union of $\mathsf{events}(t)$ for every $t \in T$. For a history h and an event e in h, $\mathsf{tr}(e)$ is the transaction t in h that contains e. We assume that each event belongs to only one transaction. Also, $\mathsf{writes}(h) = \bigcup_{t \in \mathsf{tr}(h)} \mathsf{writes}(t)$ and $\mathsf{reads}(h) = \bigcup_{t \in \mathsf{tr}(h)} \mathsf{reads}(t)$. We extend so to pairs of events by $(e_1, e_2) \in \mathsf{so}$ if $(\mathsf{tr}(e_1), \mathsf{tr}(e_2)) \in \mathsf{so}$. Also, $\mathsf{po} = \bigcup_{t \in T} \mathsf{po}_t$. We use h, h_1, h_2, ... to range over histories.

For a history h, we say that an event r *reads* x in h whenever $\mathsf{wr}_x^{-1}(r) \downarrow$. Also, we say that an event w *writes* x in h, denoted by w writes x, whenever $\texttt{value}_{\mathsf{wr}}(w, x) \neq \bot$ and the transaction of w is *not* aborted. We extend the function $\texttt{value}$ to transactions: $\texttt{value}_{\mathsf{wr}}(t, x)$ equals $\texttt{value}_{\mathsf{wr}}(w, x)$, where w is the maximal event in po_t that writes x.

2.3 Classes of histories

We define two classes of histories: (1) *full* histories which are required to define the semantics of isolation levels and (2) *client* histories which model what is observable from interacting with a database as a black-box.

Full histories model the fact that every read query "inspects" an entire snapshot of the database in order to for instance, select rows satisfying some predicate. Roughly, full histories contain a write-read dependency for every read and key. There is an exception which concerns "local" reads. If a transaction modifies a row with key x and then reads the same row, then it must always return the value written in the transaction. This holds under all isolation levels. In such


init
INSERT($\{x_1 : 0, x_2 : 1\}$)
so
wr$_{x_2}$
wr$_{x_1}$
so
t_1
UPDATE($\lambda r : r \geq 1,\ \lambda r : -2$)
t_2
DELETE($\lambda r : r \leq 0$)
init
INSERT($\{x_1 : 0, x_2 : 1\}$)
wr$_{x_1}$
so
wr$_{x_2}$
wr$_{x_1}$
so
t_1
UPDATE($\lambda r : r \geq 1,\ \lambda r : -2$)
wr$_{x_2}$
DELETE($\lambda r : r \leq 0$)
t_2
init
INSERT($\{x_1 : 0, x_2 : 1\}$)
wr$_{x_1}$
so
wr$_{x_2}$
wr$_{x_1}$
so
wr$_{x_2}$
t_1
UPDATE($\lambda r : r \geq 1,\ \lambda r : -2$)
DELETE($\lambda r : r \leq 0$)
t_2

(a) Client history.

(b) t_2 observes $x_2 = -2$.

(c) t_2 observes $x_2 = 1$.

Fig. 2: Examples of a client history h and two possible extensions. The dashed edge belongs only to the extensions. The first extension is not a witness of h as t_1 writes -2 on x_2 and $\texttt{WHERE}(t_2)(-2) = 1$.

a case, there would be no write-read dependency because these dependencies model interference across different transactions. We say that a read r reads a key x *locally* if it is preceded in the same transaction by a write w that writes x.

Definition 2. *A* full history $(T, \mathsf{so}, \mathsf{wr})$ *is a history where* $\mathsf{wr}_x^{-1}(r)$ *is defined for all* x *and* r*, unless* r *reads* x *locally.*

Client histories record less write-read dependencies compared to full histories, which is formalized by the *extends* relation.

Definition 3. *A history* $\overline{h} = (T, \mathsf{so}, \overline{\mathsf{wr}})$ extends *another history* $h = (T, \mathsf{so}, \mathsf{wr})$ *if* $\mathsf{wr} \subseteq \overline{\mathsf{wr}}$*. We denote it by* $h \subseteq \overline{h}$*.*

Definition 4. *A* client history $h = (T, \mathsf{so}, \mathsf{wr})$ *is a history s.t. there is a full history* $\overline{h} = (T, \mathsf{so}, \overline{\mathsf{wr}})$ *with* $h \subseteq \overline{h}$*, and s.t for every* x*, if* $(w, r) \in \overline{\mathsf{wr}}_x \setminus \mathsf{wr}_x$ *then* $\texttt{WHERE}(r)(\texttt{value}_{\overline{\mathsf{wr}}}(w, x)) = 0$*. The history* h' *is called a* witness *of* h*.*

Compared to a witness full history, a client history may omit write-read dependencies if the written values do *not* satisfy the predicate of the read query. These values would not be observable when interacting with the database as a black-box. This includes the case when the write is a $\texttt{DELETE}$ (recall that the special value $\dagger_x$ indicating deleted rows falsifies every predicate by convention). Figure 1 shows a full history as every query reads both x_1 and x_2. Figure 2a shows a client history: transactions t_1, t_2 does not read x_2 and x_1 resp. Figure 2b is an extension but not a witness while Figure 2c is indeed a witness of it.

3 Axiomatic Semantics With Different Isolation Levels

We define an axiomatic semantics on histories where transactions can be assigned different isolation levels, which builds on the work of Biswas et al. [7].

3.1 Executions

An *execution* of a program is represented using a history with a set of transactions T along with a total order $\mathsf{co} \subseteq T \times T$ called *commit order*. Intuitively, the commit order represents the order in which transactions are committed in the database.

Definition 5. *An* execution $\xi = (h, \mathsf{co})$ *is a history* $h = (T, \mathsf{so}, \mathsf{wr})$ *along with a* commit order $\mathsf{co} \subseteq T \times T$*, such that transactions in the same session or that are read are necessarily committed in the same order:* $\mathsf{so} \cup \mathsf{wr} \subseteq \mathsf{co}$*.* ξ *is called an execution of* h*.*

For a transaction t, we use $t \in \xi$ to denote the fact that $t \in T$. Analogously, for an event e, we use $e \in \xi$ to denote that $e \in t$ and $t \in \xi$. The extension of a commit order to pairs of events or pairs of transactions and events is done in the obvious way.

3.2 Isolation Levels

Isolation levels enforce restrictions on the commit order in an execution that depend on the session order so and the write-read relation wr. An *isolation level* ι for a transaction t is a set of constraints called *axioms*. Intuitively, an axiom states that a read event $r \in t$ reads key x from transaction t_1 if t_1 is the latest transaction that writes x which is "visible" to r – latest refers to the commit order co. Formally, an axiom a is a predicate of the following form:

$$a(r) := \forall x, t_1, t_2 . t_1 \neq t_2 \land (t_1, r) \in \mathsf{wr}_x \land t_2 \text{ } \mathsf{writes} \text{ } x \land \mathsf{vis}_a(t_2, r, x) \Rightarrow (t_2, t_1) \in \mathsf{co} \tag{1}$$

where r is a read event from t.

The visibility relation of a vis_a is described by a formula of the form:

$$\mathsf{vis}_a(\tau_0, \tau_{k+1}, x) : \exists \tau_1, \ldots, \tau_k . \bigwedge_{i=1}^{k+1} (\tau_{i-1}, \tau_i) \in \mathsf{Rel}_i \land \mathsf{WrCons}_a(\tau_0, \ldots, \tau_{k+1}, x) \tag{2}$$

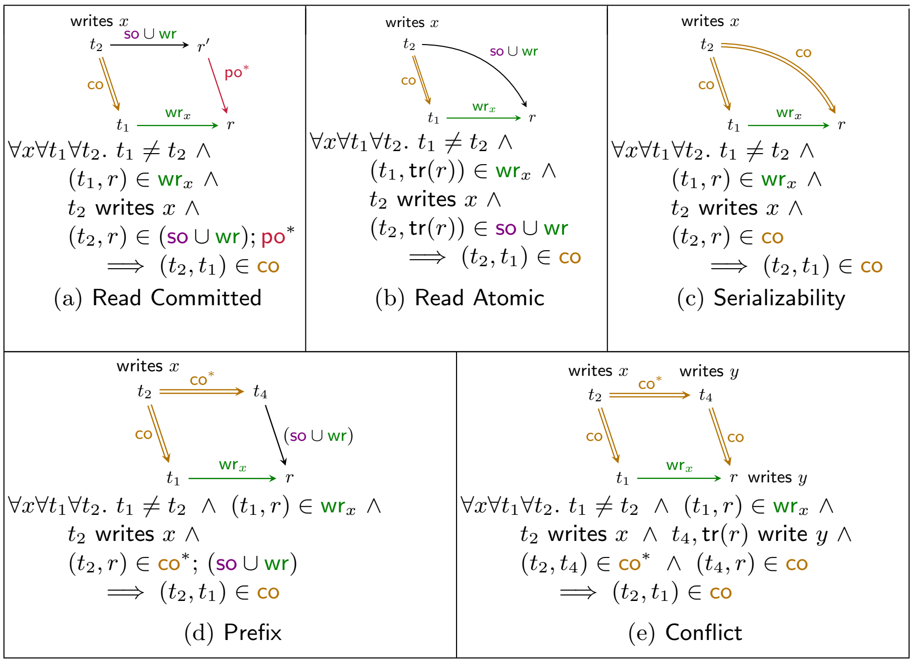
with each Rel_i is defined by the grammar:

$$\mathsf{Rel} ::= \mathsf{po} \mid \mathsf{so} \mid \mathsf{wr} \mid \mathsf{co} \mid \mathsf{Rel} \cup \mathsf{Rel} \mid \mathsf{Rel} ; \mathsf{Rel} \mid \mathsf{Rel}^+ \mid \mathsf{Rel}^* \tag{3}$$

This formula states that τ_0 (which is t_2 in Eq.1) is connected to τ_{k+1} (which is r in Eq.1) by a path of dependencies that go through some intermediate transactions or events $\tau_1, \ldots, \tau_k$. Every relation used in such a path is described based on po, so, wr and co using union $\cup$, composition of relations ;, and transitive closure operators. Finally, extra requirements on the intermediate transactions s.t. writing a different key $y \neq x$ are encapsulated in the predicate $\mathsf{WrCons}_a(\tau_0, \ldots, \tau_k, x)$.

Each axiom a uses a specific visibility relation denoted by vis_a. $\mathsf{vis}(\iota)$ denotes the set of visibility relations used in axioms defining an isolation level ι.

Figure 3 shows two axioms which correspond to their homonymous isolation levels [7]: *Read Committed* (`RC`) and *Serializability* (`SER`). `SER` states that t_2 is



$\forall x \forall t_1 \forall t_2.\ t_1 \neq t_2 \wedge$
$(t_1, r) \in \mathsf{wr}_x \wedge$
t_2 writes $x \wedge$
$(t_2, r) \in (\mathsf{so} \cup \mathsf{wr}); \mathsf{po}^*$
$\implies (t_2, t_1) \in \mathsf{co}$

(a) Read Committed

$\forall x \forall t_1 \forall t_2.\ t_1 \neq t_2 \wedge$
$(t_1, \mathsf{tr}(r)) \in \mathsf{wr}_x \wedge$
t_2 writes $x \wedge$
$(t_2, \mathsf{tr}(r)) \in \mathsf{so} \cup \mathsf{wr}$
$\implies (t_2, t_1) \in \mathsf{co}$

(b) Read Atomic

$\forall x \forall t_1 \forall t_2.\ t_1 \neq t_2 \wedge$
$(t_1, r) \in \mathsf{wr}_x \wedge$
t_2 writes $x \wedge$
$(t_2, r) \in \mathsf{co}$
$\implies (t_2, t_1) \in \mathsf{co}$

(c) Serializability

$\forall x \forall t_1 \forall t_2.\ t_1 \neq t_2 \ \wedge\ (t_1, r) \in \mathsf{wr}_x \wedge$
t_2 writes $x \wedge$
$(t_2, r) \in \mathsf{co}^*; (\mathsf{so} \cup \mathsf{wr})$
$\implies (t_2, t_1) \in \mathsf{co}$

(d) Prefix

$\forall x \forall t_1 \forall t_2.\ t_1 \neq t_2 \ \wedge\ (t_1, r) \in \mathsf{wr}_x \wedge$
t_2 writes $x \ \wedge\ t_4, \mathsf{tr}(r)$ write $y \wedge$
$(t_2, t_4) \in \mathsf{co}^* \ \wedge\ (t_4, r) \in \mathsf{co}$
$\implies (t_2, t_1) \in \mathsf{co}$

(e) Conflict

Fig. 3: Axioms defining RC, RA, SER, PC and SI isolations levels respectively. Visibility relations are "inlined" to match the definitions in [7].

visible to r if t_2 commits before r, while RC states that t_2 is visible to r if either $(t_2, r) \in \mathsf{so}$ or if there exists a previous event r' in $\mathsf{tr}(r)$ that reads x from t_2. Similarly, *Read Atomic* (RA) and *Prefix Consistency* (PC) are defined using their homonymous axioms while *Snapshot Isolation* (SI) is defined as a conjunction of both Prefix and Conflict.

The *isolation configuration* of a history is a mapping $\mathsf{iso}(h) : T \to \mathsf{Iso}$ associating to each transaction an isolation level identifier from a set Iso.

Whenever every transaction in a history has the same isolation level ι, the isolation configuration of that history is denoted simply by ι.

Note that SER is stronger than RC: every transaction visible to a read r according to RC is also visible to r according to SER. This means SER imposes more constraints for transaction t_1 to be read by r than RC. In general, for two isolation configurations I_1 and I_2, I_1 is *stronger than* I_2 when for every transaction t, $I_1(t)$ is stronger than $I_2(t)$ (i.e., whenever $I_1(t)$ holds in an execution ξ, $I_2(t)$ also holds in ξ). The *weaker than* relationship is defined similarly.

Given a history h with isolation configuration $\mathsf{iso}(h)$, h is called *consistent* when there exists an execution ξ of h such that for all transactions t in ξ, the axioms in $\mathsf{iso}(h)(t)$ are satisfied in ξ (the interpretation of an axiom over an execution is defined as expected). For example, let h be the full history in Figure 2c. If both t_1, t_2's isolation are SER, then h is *not* consistent, i.e., every execution $\xi = (h, \mathsf{co})$ violates the corresponding axioms. Assume for instance, that $(t_1, t_2) \in \mathsf{co}$. Then, by axiom SER, as $(\mathtt{init}, t_2) \in \mathsf{wr}_{x_1}$ and t_1 writes x_1, we get that $(t_1, \texttt{init}) \in \mathsf{co}$, which is impossible as $(\texttt{init}, t_1) \in \mathsf{so} \subseteq \mathsf{co}$. However, if the isolation configuration is weaker (for example $\mathsf{iso}(h)(t_2) = \texttt{RC}$), then the history is consistent using $\texttt{init} <_{\mathsf{co}} t_1 <_{\mathsf{co}} t_2$ as commit order.

Definition 6. *A full history $h = (T, \mathsf{so}, \mathsf{wr})$ with isolation configuration $\mathsf{iso}(h)$ is* consistent *iff there is an execution ξ of h s.t. $\bigwedge_{t \in T, r \in \mathsf{reads}(t), a \in \mathsf{iso}(h)(t)} a(r)$ holds in ξ; ξ is called a* consistent *execution of h.*

The notion of consistency on full histories is extended to client histories.

Definition 7. *A client history $h = (T, \mathsf{so}, \mathsf{wr})$ with isolation configuration $\mathsf{iso}(h)$ is* consistent *iff there is a full history $\overline{h}$ with the same isolation configuration which is a witness of h and consistent; $\overline{h}$ is called a* consistent *witness of h.*

In general, the witness of a client history may not be consistent. In particular, there may exist several witnesses but no consistent witness.

3.3 Validation of the semantics

To justify the axiomatic semantics defined above, we define an operational semantics inspired by real implementations and prove that every run of a program can be translated into a consistent history. Every instruction is associated with an increasing timestamp and it reads from a snapshot of the database defined according to the isolation level of the enclosing transaction. At the end of the transaction we evaluate if the transaction can be committed or not. We assume that a transaction can abort only if explicitly stated in the program. We model an optimistic approach where if a transaction cannot commit, the run blocks (modelling unexpected aborts). We focus on three of the most used isolation levels: $\texttt{SER}, \texttt{SI}, \texttt{RC}$. Other isolation levels can be handled in a similar manner. For each run ρ we extract a full history $\mathsf{history}(\rho)$. We show by induction that $\mathsf{history}(\rho)$ is consistent at every step.

Theorem 1. *For every run ρ, $\mathsf{history}(\rho)$ is consistent.*

4 Complexity of Checking Consistency

4.1 Saturation and Boundedness

We investigate the complexity of checking if a history is consistent. Our axiomatic framework characterize isolation levels as a conjunction of axioms as in Equation (1). However, some isolation levels impose stronger constraints than others. For studying the complexity of checking consistency, we classify them in two categories, saturable or not. An isolation level is *saturable* if its visibility relations are defined without using the co relation (i.e. the grammar in Equation (3) omits the co relation). Otherwise, we say that the isolation level is *non-saturable*. For example, $\texttt{RC}$ and $\texttt{RA}$ are saturable while $\texttt{PC}, \texttt{SI}$ and $\texttt{SER}$ are not.

Algorithm 1 Extending an initial pco relation with necessary ordering constraints

```
function SATURATE($h = (T, \mathsf{so}, \mathsf{wr})$, pco)        ▷ pco must be transitive.
    $\mathsf{pco}_{\mathtt{res}} \leftarrow \mathsf{pco}$
    for all $x \in \mathsf{Keys}$ do
        for all $r \in \mathsf{reads}(h), t_2 \neq \mathsf{tr}(r) \in T$ s.t. $t_2$ writes $x$ and $t_2 \neq \mathsf{tr}(\mathsf{wr}_x^{-1}(r))$ do
            $t_1 \leftarrow \mathsf{tr}(\mathsf{wr}_x^{-1}(r))$        ▷ $t_1$ is well defined as $h$ is a full history.
            for all $\mathsf{v} \in \mathsf{vis}(\mathsf{iso}(h)(\mathsf{tr}(r)))$ do
                if $\mathsf{v}(t_2, r, x)$ then
                    $\mathsf{pco}_{\mathtt{res}} \leftarrow \mathsf{pco}_{\mathtt{res}} \cup \{(t_2, t_1)\}$
    return $\mathsf{pco}_{\mathtt{res}}$
```

Algorithm 2 Checking saturable consistency

```
function CHECKSATURABLE($h = (T, \mathsf{so}, \mathsf{wr})$)
    if $\mathsf{so} \cup \mathsf{wr}$ is cyclic then return false
    $\mathsf{pco} \leftarrow$ SATURATE($h, (\mathsf{so} \cup \mathsf{wr})^+$)
    return true if pco is acyclic, and false, otherwise
```

Definition 8. *An isolation configuration* $\mathsf{iso}(h)$ *is* saturable *if for every transaction* t*,* $\mathsf{iso}(h)(t)$ *is a saturable isolation level. Otherwise,* $\mathsf{iso}(h)$ *is* non-saturable*.*

We say an isolation configuration $\mathsf{iso}(h)$ is *bounded* if there exists a fixed $k \in \mathbb{N}$ s.t. for every transaction t, $\mathsf{iso}(h)(t)$ is defined as a conjunction of at most k axioms that contain at most k quantifiers. For example, `SER` employs one axiom and four quantifiers while `SI` employs two axioms, Prefix and Conflict, with four and five quantifiers respectively. Any isolation configuration composed with `SER`, `SI`, `PC`, `RA` and `RC` isolation levels is bounded. We assume in the following that isolation configurations are bounded.

Checking consistency requires computing the $\mathtt{value}_{\mathsf{wr}}$ function and thus, evaluating `WHERE` predicates. In the following, we assume that evaluating `WHERE` predicates on a single row requires constant time.

4.2 Checking Consistency of Full Histories

Algorithm 2 computes necessary and sufficient conditions for the existence of a consistent execution $\xi = (h, \mathsf{co})$ for a history h with a saturable isolation configuration. It calls SATURATE, defined in Algorithm 1, to compute a "*partial*" commit order relation pco that includes $(\mathsf{so} \cup \mathsf{wr})^+$ and any other dependency between transactions that can be deduced from the isolation configuration. A consistent execution exists iff this partial commit order is acyclic. Algorithm 2 generalizes the results in [7] for full histories with heterogeneous saturable isolation configurations.

Theorem 2. *Checking consistency of full histories with bounded saturable isolation configurations can be done in polynomial time.*

For bounded non-saturable isolation configurations, checking if a history is consistent is NP-complete as an immediate consequence of the results in [7]. These previous results apply to the particular case of transactions having the same isolation level and being formed of classic read and write instructions on a fixed set of variables. The latter can be simulated by SQL queries using WHERE predicates for selecting rows based on their key being equal to some particular value. For instance, $\texttt{SELECT}(\lambda r : \mathsf{key}(r) = x)$ simulates a read of a "variable" x.

4.3 Checking Consistency of Client Histories

We show that going from full histories to client histories, the consistency checking problem becomes NP-complete, independently of the isolation configurations. Intuitively, NP-hardness comes from keys that are not included in outputs of SQL queries. The justification for the consistency of omitting such rows can be ambiguous, e.g., multiple values written to a row may not satisfy the predicate of the WHERE clause, or multiple deletes can justify the absence of a row.

The *width* of a history $\texttt{width}(h)$ is the maximum number of transactions which are pairwise incomparable w.r.t. so. In a different context, previous work [7] showed that bounding the width of a history (consider it to be a constant) is a sufficient condition for obtaining polynomial-time consistency checking algorithms. This is not true for client histories.

Theorem 3. *Checking consistency of bounded-width client histories with bounded isolation configuration stronger than* RC *and* $\texttt{width}(h) \geq 3$ *is NP-complete.*

The proof of NP-hardness uses a reduction from 1-in-3 SAT which is inspired by the work of Gibbons and Korach [16] (Theorem 2.7) concerning sequential consistency for shared memory implementations. Our reduction is a non-trivial extension because it has to deal with any weak isolation configuration stronger than RC.

The proof of Theorem 3 relies on using non-trivial predicates in WHERE clauses. We also prove that checking consistency of client histories is NP-complete irrespectively of the complexity of these predicates. This result uses another class of histories, called *partial-observation* histories. These histories are a particular class of client histories where events read all inserted keys, irrespectively of their WHERE clauses (as if these clauses where *true*).

Definition 9. *A* partial observation history $h = (T, \mathsf{so}, \mathsf{wr})$ *is a client history for which there is a witness* $\overline{h} = (T, \mathsf{so}, \overline{\mathsf{wr}})$ *of* h*, s.t. for every* x*, if* $(w, r) \in \overline{\mathsf{wr}}_x \setminus \mathsf{wr}_x$*, then* w *deletes* x*.*

Theorem 4. *Checking consistency of partial observation histories with bounded isolation configurations stronger than* RC *is NP-complete.*

The proof of NP-hardnessuses a novel reduction from 3 SAT. The main difficulty for obtaining consistent witnesses of partial observation histories is the ambiguity of which delete event is responsible for each absent row.

Algorithm 3 Checking consistency of client histories

```
 1: function CHECKCONSISTENCY($h = (T, \mathsf{so}, \mathsf{wr})$)
 2:   let $\mathsf{pco} = \mathsf{FIX}(\lambda R : \text{SATURATE}(h, R))(\mathsf{so} \cup \mathsf{wr})^+$
 3:   let $E_h = \{(r, x) \mid r \in \mathsf{reads}(h), x \in \mathsf{Keys}.\mathsf{wr}_x^{-1}(r) \uparrow$ and $\mathtt{1}_r^x(\mathsf{pco}) \neq \emptyset\}$
 4:   let $X_h$ = the set of mappings that map each $(r, x) \in E_h$ to a member of $\mathtt{0}_x^r(\mathsf{pco})$
 5:   if $\mathsf{pco}$ is cyclic then return false
 6:   else if there exists $(r, x) \in E_h$ such that $\mathtt{0}_x^r(\mathsf{pco}) = \emptyset$ then return false
 7:   else if $E_h = \emptyset$ then return EXPLORECONSISTENTPREFIXES($h, \emptyset$)
 8:   else
 9:     for all $f \in X_h$ do
10:       seen $\leftarrow \emptyset$; $h' \leftarrow h \bigoplus_{(r,x) \in E_h} \mathsf{wr}_x(f(r, x), r)$
11:       if EXPLORECONSISTENTPREFIXES($h', \emptyset$) then return true
12:     return false
```

5 Effectively Checking Consistency of Client Histories

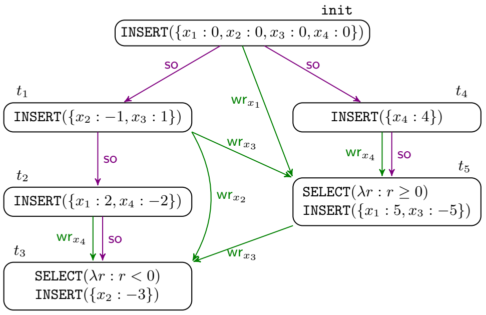
The result of Theorem 3 implicitly asks whether there exist conditions on the histories for which checking consistency remains polynomial as in [7]. We describe an algorithm for checking consistency of client histories and identify cases in which it runs in polynomial time.

Consider a client history $h = (T, \mathsf{so}, \mathsf{wr})$ which is consistent. For every consistent witness $\overline{h} = (T, \mathsf{so}, \overline{\mathsf{wr}})$ of h there exists a consistent execution of $\overline{h}$, $\xi = (\overline{h}, \mathsf{co})$. The commit order co contains $(\mathsf{so} \cup \mathsf{wr})^+$ and any other ordering constraint derived from axioms by observing that $(\mathsf{so} \cup \mathsf{wr})^+ \subseteq \mathsf{co}$. More generally, co includes all constraints generated by the least fixpoint of the function SATURATE defined in Algorithm 1 when starting from $(\mathsf{so} \cup \mathsf{wr})^+$ as partial commit order. This least fixpoint exists because SATURATE is monotonic. It is computed as usual by iterating SATURATE until the output does not change. We use $\mathsf{FIX}(\lambda R : \text{SATURATE}(h, R))(\mathsf{so} \cup \mathsf{wr})^+$ to denote this least fixpoint. In general, such a fixpoint computation is just an under-approximation of co, and it is not enough for determining h's consistency.

The algorithm we propose, described in Algorithm 3, exploits the partial commit order pco obtained by such a fixpoint computation (line 2) for determining h's consistency. For a read r, key x, we define $\mathtt{1}_x^r(\mathsf{pco})$, resp., $\mathtt{0}_x^r(\mathsf{pco})$, to be the set of transactions that are *not* committed after $\mathtt{tr}(r)$ and which write a value that satisfies, resp., does not satisfy, the predicate $\mathtt{WHERE}(r)$. The formal description of both sets can be seen in Equation 4.

$$\mathtt{1}_x^r(\mathsf{pco}) = \{t \in T \mid (\mathtt{tr}(r), t) \notin \mathsf{pco} \ \wedge \ \mathtt{WHERE}(r)(\mathtt{value}_{\mathsf{wr}}(t, x)) = 1\}$$
$$\mathtt{0}_x^r(\mathsf{pco}) = \{t \in T \mid (\mathtt{tr}(r), t) \notin \mathsf{pco} \ \wedge \ \mathtt{WHERE}(r)(\mathtt{value}_{\mathsf{wr}}(t, x)) = 0\} \qquad (4)$$

The set $\mathtt{0}_x^r(\mathsf{pco})$ can be used to identify extensions that are not witness of a history. Let us consider the client history h depicted in Figure 4a. Observe that t_3 is not reading x_1 and t_5 is not reading x_2. Table 4b describes all possible full extensions $\overline{h}$ of h. An execution $\xi = (\overline{h}, \mathsf{co})$ is consistent if $(t, r) \in \overline{\mathsf{wr}}_x \setminus \mathsf{wr}_x$ implies $\mathtt{WHERE}(r)(\mathtt{value}_{\mathsf{wr}}(t, x)) = 0$. This implies that extensions h_1, h_4, and h_7, where


init
INSERT($\{x_1 : 0, x_2 : 0, x_3 : 0, x_4 : 0\}$)
so
so
wr_{x_1}
t_1
INSERT($\{x_2 : -1, x_3 : 1\}$)
t_4
INSERT($\{x_4 : 4\}$)
so
wr_{x_3}
wr_{x_4}
so
t_5
t_2
INSERT($\{x_1 : 2, x_4 : -2\}$)
wr_{x_2}
SELECT($\lambda r : r \geq 0$)
INSERT($\{x_1 : 5, x_3 : -5\}$)
wr_{x_4}
so
t_3
wr_{x_3}
SELECT($\lambda r : r < 0$)
INSERT($\{x_2 : -3\}$)

(a) A history where t_3, t_5 have PC and SER as isolation levels respectively. The isolation levels of the other transactions are unspecified.

History	$\mathsf{wr}_{x_1}^{-1}(t_3)$	$\mathsf{wr}_{x_2}^{-1}(t_5)$
h_1	init	init
h_2	init	t_1
h_3	init	t_3
h_4	t_2	init
h_5	t_2	t_1
h_6	t_2	t_3
h_7	t_5	init
h_8	t_5	t_1
h_9	t_5	t_3

(b) Table describing all possible full extensions of the history in Figure 4a.

History	$\mathsf{wr}_{x_1}^{-1}(t_3)$	$\mathsf{wr}_{x_2}^{-1}(t_5)$
h_{258}	**undef**	t_1

(c) Table describing the only conflict-free extension of Figure 4a.

Fig. 4: Comparison between conflict-free extensions and full extensions of the history h in Figure 4a. In h, wr^{-1} is not defined for two pairs: (t_3, x_1) and (t_5, x_2); where we identify the single SELECT event in a transaction with its transaction. Table 4b describes all possible full extensions of h. For example, the first extension, h_1, states that $(\mathtt{init}, t_3) \in \mathsf{wr}_{x_1}$ and $(\mathtt{init}, t_5) \in \mathsf{wr}_{x_2}$. Algorithm 3 only explore the only extension h_{258} described in Table 4c; where $\mathsf{wr}_{x_1}^{-1}(t_3) \uparrow$ and $(t_1, t_5) \in \mathsf{wr}_{x_2}$. The history h_{258} can be extended to histories h_2, h_5 and h_8.

$(\mathtt{init}, t_5) \in \overline{\mathsf{wr}}_{x_2}$, are not witnesses of h as $\mathtt{WHERE}(t_5)(\mathtt{value}_{\mathsf{wr}}(\mathtt{init}, x_2)) = 1$. We note that $\mathtt{init} \notin \mathtt{0}_{x_2}^{t_5}(\mathsf{pco}) = \{t_1\}$. Also, observe that $(t_5, t_3) \in \mathsf{wr}$; so extensions h_3, h_6 and h_9, where $(t_3, t_5) \in \overline{\mathsf{wr}}_{x_2}$, are not a witness of h. Once again, $t_3 \notin \mathtt{0}_{x_2}^{t_5}(\mathsf{pco})$. In general, for every read event r and key x s.t. $\mathsf{wr}_x^{-1}(r) \uparrow$, the extension of h where $(t, r) \in \overline{\mathsf{wr}}_x$, $t \notin \mathtt{0}_x^r(\mathsf{pco})$, is not a witness of h. In particular, if $\mathsf{wr}_x^{-1}(r) \uparrow$ but $\mathtt{0}_x^r(\mathsf{pco}) = \emptyset$, then no witness of h can exist.

The sets $\mathtt{0}_x^r(\mathsf{pco})$ are not sufficient to determine if a witness is a consistent witness as our previous example shows: $\mathtt{0}_{x_1}^{t_3}(\mathsf{pco}) = \{\mathtt{init}, t_2, t_5\}$, but h_2 is not consistent. Algorithm 3, combines an enumeration of history extensions with a search for a consistent execution of each extension. The extensions are *not* necessarily full. In case $\mathsf{wr}_x^{-1}(r)$ is undefined, we use sets $\mathtt{1}_x^r(\mathsf{pco})$ to decide whether the extension of h requires specifying $\mathsf{wr}_x^{-1}(r)$ for determining h's consistency. Algorithm 3 specifies $\mathsf{wr}_x^{-1}(r)$ only if (r, x) is a so-called *conflict*, i.e., $\mathsf{wr}_x^{-1}(r)$ is undefined and $\mathtt{1}_x^r(\mathsf{pco}) \neq \emptyset$.

Following the example of Figure 4, we observe that $\mathtt{1}_{x_1}^{t_3}(\mathsf{pco}) = \emptyset$, all transactions that write on x_1 write non-negative values; but instead $\mathtt{1}_{x_2}^{t_5}(\mathsf{pco}) = \{\mathtt{init}\}$. Intuitively, this means that if some extension h' that does not specify $\mathsf{wr}_{x_1}^{-1}(t_3)$ does not violate any axiom when using some commit order co, then we can extend h', defining $\mathsf{wr}_{x_1}^{-1}(t_3)$ as some adequate transaction, and obtain a full history $\overline{h}$ s.t. the execution $\xi = (\overline{h}, \mathsf{co})$ is consistent. On the other hand, specifying the write-read dependency of t_5 on x_2 matters. For not contradicting any axiom using co, we may require $(\texttt{init}, t_5) \in \overline{\mathsf{wr}}_{x_2}$. However, such extension is not even a witness of h as $\texttt{WHERE}(\texttt{init})(\texttt{value}_{\mathsf{wr}}(\texttt{init}, x_2)) = 1$. This intuition holds for the particular definitions of the isolation levels that Algorithm 3 considers.

A history is *conflict-free* if it does not have conflicts. Our previous discussion reduces the problem of checking consistency of a history to checking consistency of its conflict-free extensions. For example, the history h in Figure 4a is not conflict-free but the extension h_{258} defined in Table 4c is. Instead of checking consistency of the nine possible extensions, we only check consistency of h_{258}.

Algorithm 3 starts by checking if there is at least a conflict-free extension of h (line 6). If h is conflict-free, it directly calls Algorithm 4 (line 7); while otherwise, it iterates over conflict-free extensions of h, calling Algorithm 4 on each of them (line 11).

Algorithm 4 describes the search for the commit order of a conflict-free history h. This is a recursive enumeration of consistent prefixes of histories that backtracks when detecting inconsistency (it generalizes Algorithm 2 in [7]). A *prefix* of a history $h = (T, \mathsf{so}, \mathsf{wr})$ is a tuple $P = (T_P, M_P)$ where $T_P \subseteq T$ is a set of transactions and $M_P : \mathsf{Keys} \to T_P$ is a mapping s.t. (1) so predecessors of transactions in T_P are also in T_P, i.e., $\forall t \in T_P.\ \mathsf{so}^{-1}(t) \in T_P$ and (2) for every x, $M_P(x)$ is a so-maximal transaction in T_P that writes x (M_P records a last write for every key).

For every prefix $P = (T_P, M_P)$ of a history h and a transaction $t \in T \setminus T_P$, we say a prefix $P' = (T_{P'}, M_{P'})$ of h is an *extension* of P *using* t if $T_{P'} = T_P \cup \{t\}$ and for every key x, $M_{P'}(x)$ is t or $M_P(x)$. Algorithm 4 extensions, denoted as $P \cup \{t\}$, guarantee that for every key x, if t writes x, then $M_{P'}(x) = t$.

Extending the prefix P using t means that any transaction $t' \in T_P$ is committed before t. Algorithm 4 focuses on special extensions that lead to commit orders of consistent executions.

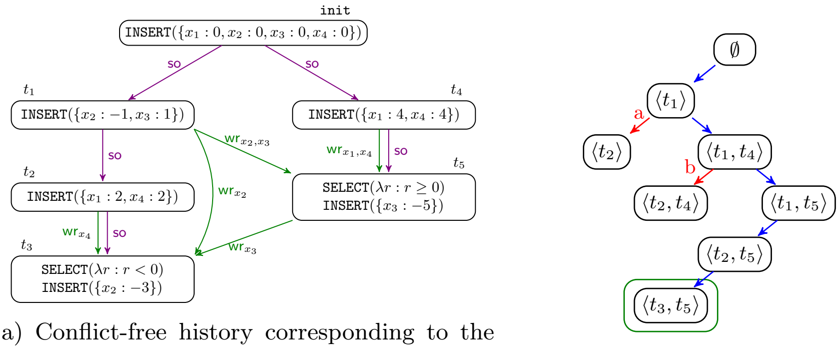
Axiom	**Predicate**
Serializability, Prefix, Read Atomic, Read Committed	$\nexists x \in \mathsf{Keys}$ s.t. t writes x, $\mathsf{wr}_x^{-1}(r) \downarrow$ $v(\mathsf{pco}_t^P)(t, r, x)$ holds in h and $\mathsf{wr}_x^{-1}(r) \in T_P$
Conflict	$\nexists x \in \mathsf{Keys}, t' \in T_P \cup \{t\}$ s.t. t' writes x, $\mathsf{wr}_x^{-1}(r) \downarrow$ $v(\mathsf{pco}_t^P)(t', r, x)$ holds in h and $\mathsf{wr}_x^{-1}(r) \neq M_P(x)$

Table 1: Predicates relating prefixes and visibility relations where pco_t^P is defined as $\mathsf{pco} \cup \{(t', t) \mid t' \in T_P\} \cup \{(t, t'') \mid t'' \in T \setminus (T_P \cup \{t\})\}$.

Definition 10. *Let h be a history, $P = (T_P, M_P)$ be a prefix of h, t a transaction that is not in T_P and $P' = (T_{P'}, M_{P'})$ be an exetension of P using t. The prefix P' is a* consistent extension *of P with t, denoted by $P \triangleright_t P'$, if*

1. *P is pco-closed: for every transaction $t' \in T$ s.t. $(t', t) \in \mathsf{pco}$ then $t' \in T_P$,*
2. *t does not overwrite other transactions in P: for every $\texttt{read}$ event r outside of the prefix, i.e., $\mathsf{tr}(r) \in T \setminus T_{P'}$ and every visibility relation $v \in \mathsf{vis}(\mathsf{iso}(h))(\mathsf{tr}(r))$, the predicate $\mathsf{vp}_v^P(t, r)$ defined in Table 1 holds in h.*

We say that a prefix is consistent if it is either the empty prefix or it is a consistent extension of a consistent prefix.


init
INSERT({$x_1: 0, x_2: 0, x_3: 0, x_4: 0$})
so
so
t_1
INSERT({$x_2: -1, x_3: 1$})
t_4
INSERT({$x_1: 4, x_4: 4$})
so
wr_{x_2,x_3}
wr_{x_1,x_4}
so
t_5
t_2
INSERT({$x_1: 2, x_4: 2$})
wr_{x_2}
SELECT($\lambda r: r \geq 0$)
INSERT({$x_3: -5$})
wr_{x_4}
so
t_3
wr_{x_3}
SELECT($\lambda r: r < 0$)
INSERT({$x_2: -3$})
$\emptyset$
$\langle t_1 \rangle$
a
$\langle t_2 \rangle$
$\langle t_1, t_4 \rangle$
b
$\langle t_2, t_4 \rangle$
$\langle t_1, t_5 \rangle$
$\langle t_2, t_5 \rangle$
$\langle t_3, t_5 \rangle$

(a) Conflict-free history corresponding to the extension h_{258} (Table 4c) of the history in Figure 4a

(b) Execution of Algorithm 3 on the history in Figure 5a.

Fig. 5: Applying Algorithm 4 on the conflict-free consistent history h_{258} on the left. The right part pictures a search for valid extensions of consistent prefixes on h_{258}. Prefixes are represented by their so-maximal transactions, e.g., $\langle t_2 \rangle$ contains all transactions which are before t_2 in so, i.e., $\{\mathtt{init}, t_1, t_2\}$. A red arrow means that the search is blocked (the prefix at the target is not a consistent extension), while a blue arrow mean that the search continues.

Figure 5b depicts the execution of Algorithm 4 on the conflict-free history Figure 5a (history h_{258} from Table 4c). Blocked and effectuated calls are represented by read and blue arrows respectively. The read arrow a is due to condition 1 in Definition 10: as t_3 enforces $\mathtt{PC}$, reads x_4 from t_2, and t_4 is visible to it ($\mathsf{vis}_{\mathsf{Prefix}}(t_4, t_3, x_4)$), $(t_4, t_2) \in \mathsf{pco}$; so consistent prefixes can not contain t_2 if they do not contain t_4. The read arrow b is due to condition 2: as t_5 enforces $\mathtt{SER}$ and it reads x_1 from t_4, consistent prefixes can not contain t_2 unless t_5 is included. When reaching prefix $\langle t_3, t_5 \rangle$, the search terminates and deduces that h is consistent. From the commit order induced by the search tree we can construct the extension of h where missing write-read dependencies are obtained by applying the axioms on such a commit order. In our case, from $\mathtt{init} <_{\mathsf{co}} t_1 <_{\mathsf{co}} t_4 <_{\mathsf{co}} t_5 <_{\mathsf{co}} t_2 <_{\mathsf{co}} t_3$, we deduce that the execution $\xi = (h_5, \mathsf{co})$ is a consistent execution of h_{258}, and hence of h; where h_5 is the history described in Table 4b.

For complexity optimizations, Algorithm 4 requires an isolation level-dependent equivalence relation between consistent prefixes. If there is transaction $t \in T$ s.t. $\mathsf{iso}(h)(t) = \mathtt{SI}$, prefixes $P = (T_P, M_P)$ and $P' = (T_{P'}, M_{P'})$ are *equivalent* iff they are equal (i.e. $T_P = T_{P'}, M_P = M_{P'}$). Otherwise, they are *equivalent* iff $T_P = T_{P'}$.

Theorem 5. *Let h be a client history whose isolation configuration is defined using* $\{\mathtt{SER}, \mathtt{SI}, \mathtt{PC}, \mathtt{RA}, \mathtt{RC}\}$. *Algorithm 3 returns* $\mathtt{true}$ *if and only if h is consistent.*

In general, Algorithm 3 is exponential the number of conflicts in h. The number of *conflicts* is denoted by $\#\mathtt{conf}(h)$. The number of conflicts exponent is implied by the number of mappings in X_h explored by Algorithm 3 (E_h is the set of conflicts in h). The history width and size exponents comes from the

Algorithm 4 check consistency of conflict-free histories

```
function exploreConsistentPrefixes($h = (T, \mathsf{so}, \mathsf{wr}), P$)
  if $|P| = |T|$ then return true
  for all $t \in T \setminus P$ s.t. $P \triangleright_t (P \cup \{t\})$ do
    if $\exists P' \in \mathsf{seen}$ s.t. $P' \equiv_{\mathsf{iso}(h)} (P \cup \{t\})$ then continue
    else if exploreConsistentPrefixes($h, P \cup \{t\}$) then return true
    else $\mathsf{seen} \leftarrow \mathsf{seen} \cup (P \cup \{t\})$
  return false
```

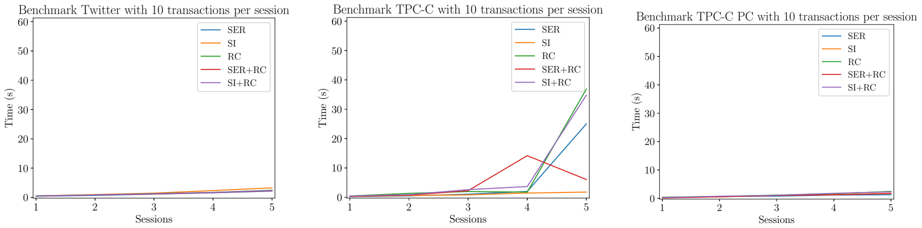


Fig. 6: Running time of Algorithm 3 while increasing the number of sessions. Each point represents the average running time of 5 random clients of such size.

number of prefixes explored by Algorithm 4 which is $|h|^{\mathtt{width}(h)} \cdot \mathtt{width}(h)^{|\mathsf{Keys}|}$ in the worst case (prefixes can be equivalently described by a set of so-maximal transactions and a mapping associating keys to sessions).

Theorem 6. *For every client history h whose isolation configuration is composed of $\{\mathtt{SER}, \mathtt{SI}, \mathtt{PC}, \mathtt{RA}, \mathtt{RC}\}$ isolation levels, Algorithm 3 runs in $\mathcal{O}(|h|^{\#\mathtt{conf}(h)+\mathtt{width}(h)+9} \cdot \mathtt{width}(h)^{|\mathsf{Keys}|})$. Moreover, if no transaction employs $\mathtt{SI}$ isolation level, Algorithm 3 runs in $\mathcal{O}(|h|^{\#\mathtt{conf}(h)+\mathtt{width}(h)+8})$.*

On bounded, conflict-free histories only using $\mathtt{SER}, \mathtt{PC}, \mathtt{RA}, \mathtt{RC}$ as isolation levels, Algorithm 3 runs in polynomial time. For instance, standard reads and writes can be simulated using `INSERT` and `SELECT` with `WHERE` clauses that select rows based on their key being equal to some particular value. In this case, histories are conflict-less (wr would be defined for the particular key asked by the clause, and writes on other keys would not satisfy the clause). A more general setting where `WHERE` clauses restrict only values that are immutable during the execution (e.g., primary keys) and deletes only affect non-read rows also falls in this category.

6 Experimental evaluation

We evaluate an implementation of checkConsistency in the context of the Benchbase [12] database benchmarking framework. We apply this algorithm on histories extracted from randomly generated client programs of a number of database-backed applications. We use PostgreSQL 14.10 as a database. The experiments were performed on an Apple M1 with 8 cores and 16 GB of RAM.

Implementation. We extend the Benchbase framework with an additional package for generating histories and checking consistency. Applications from

Benchbase are instrumented in order to be able to extract histories, the wr relation in particular. Our implementation is publicly available [5].

Our tool takes as input a configuration file specifying the name of the application and the isolation level of each transaction in that application. For computing the wr relation and generating client histories, we extend the database tables with an extra column `WRITEID` which is updated by every `write` instruction with a unique value. SQL queries are also modified to return whole rows instead of selected columns. To extract the wr relation for `UPDATE` and `DELETE` we add `RETURNING` clauses. Complex operators such as `INNER JOIN` are substituted by simple juxtaposed SQL queries (similarly to [8]). We map the result of each query to local structures for generating the corresponding history. Transactions aborted by the database (and not explicitly by the application) are discarded.

Benchmark. We analyze a set of benchmarks inspired by real-world applications and evaluate them under different types of clients and isolation configurations. We focus on isolation configurations implemented in PostgreSQL, i.e. compositions of $\mathtt{SER}, \mathtt{SI}$ and $\mathtt{RC}$ isolation levels.

In average, the ratio of SER/SI transactions is 11% for Twitter and 88% for TPC-C and TPC-C PC. These distributions are obtained via the random generation of client programs implemented in BenchBase. In general, we observe that the bottleneck is the number of possible history extensions enumerated at line 9 in Alg. 3 and not the isolation configuration. This number is influenced by the distribution of types of transactions, e.g., for TPC-C, a bigger number of transactions creating new orders increases the number of possible full history extensions. We will clarify.

Twitter [12] models a social network that allows users to publish tweets and get their followers, tweets and tweets published by other followers. We consider five isolation configurations: $\mathtt{SER}$, $\mathtt{SI}$ and $\mathtt{RC}$ and the heterogeneous $\mathtt{SER}+\mathtt{RC}$ and $\mathtt{SI}+\mathtt{RC}$, where publishing a tweet is $\mathtt{SER}$ (resp., $\mathtt{SI}$) and the rest are $\mathtt{RC}$. The ratio of $\mathtt{SER}$ (resp. $\mathtt{SI}$) transactions w.r.t. $\mathtt{RC}$ is 11% on average.

TPC-C [24] models an online shopping application with five types of transactions: reading the stock, creating a new order, getting its status, paying it and delivering it. We consider five isolation configurations: the homogeneous $\mathtt{SER}$, $\mathtt{SI}$ and $\mathtt{RC}$ and the combinations $\mathtt{SER}+\mathtt{RC}$ and $\mathtt{SI}+\mathtt{RC}$, where creating a new order and paying it have $\mathtt{SER}$ (respectively $\mathtt{SI}$) as isolation level while the rest have $\mathtt{RC}$. The ratio of $\mathtt{SER}$ (resp. $\mathtt{SI}$) transactions w.r.t. $\mathtt{RC}$ is 88% on average.

TPC-C PC is a variant of the TPC-C benchmark whose histories are always conflict-free. `DELETE` queries are replaced by `UPDATE` with the aid of extra columns simulating the absence of a row. Queries whose `WHERE` clauses query mutable values are replaced by multiple simple instructions querying only immutable values such as unique ids and primary keys.

Experimental Results. We designed two experiments to evaluate checkConsistency's performance for different isolation configurations increasing the number of transactions per session (the number of sessions is fixed), the number of sessions (the number of transactions per session is fixed), resp. We use a timeout of 60 seconds per history.

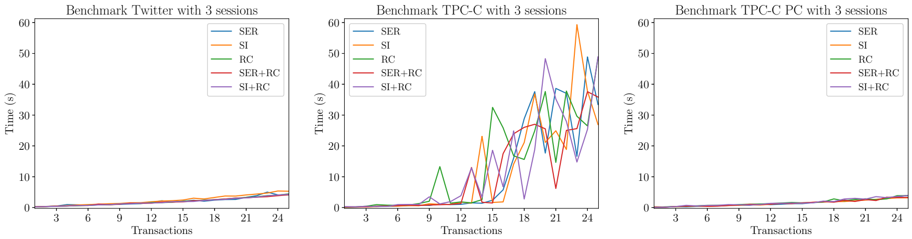


Fig. 7: Running time of Algorithm 3 increasing the number of transactions per session. We plot the average running time of 5 random clients of such size.

The first experiment investigates the scalability of Algorithm 3 when increasing the number of sessions. For each benchmark and isolation configuration, we consider 5 histories of random clients (each history is for a different client) with an increasing number of sessions and 10 transactions per session (around 400 histories across all benchmarks). No timeouts appear with less than 4 sessions. Figure 6 shows the running time of the experiment.

The second experiment investigates the scalability of Algorithm 3 when increasing the number of transactions. For each benchmark and isolation configuration, we consider 5 histories of random clients, each having 3 sessions and an increasing number of transactions per session (around 1900 histories across all benchmarks). Figure 7 shows its running time.

The runtime similarities between isolation configurations containing `SI` versus those without it show that in practice, the bottleneck of Algorithm 3 is the number of possible history extensions enumerated at line 11 in Algorithm 3; i.e. the number of conflicts in a history. This number is influenced by the distribution of types of transactions, e.g., for TPC-C, a bigger number of transactions creating new orders increases the number of possible full history extensions. Other isolation levels not implemented by PostgreSQL, e.g., prefix consistency `PC`, are expected to produce similar results.

Both experiments show that Algorithm 3 scales well for histories with a small number of writes (like Twitter) or conflicts (like TPC-C PC). In particular, Algorithm 3 is quite efficient for typical workloads needed to expose bugs in production databases which contain less than 10 transactions [7,20,18].

A third experiment compares Algorithm 3 with a baseline consisting in a naive approach where we enumerate witnesses and executions of such witnesses until consistency is determined. We consider Twitter and TPC-C as benchmarks and execute 5 histories of random clients, each having 3 sessions and an increasing number of transactions per session (around 100 histories across all benchmarks). We execute each client under `RC` and check the obtained histories for consistency with respect to `SER`.

The naive approach either times out for 35.5%, resp., 95.5% of the histories of Twitter, resp., TPC-C, or finishes in 5s on average (max 25s). In comparison, Algorithm 3 has no timeouts for Twitter and times out for 5.5% of the TPC-C histories; finishing in 1.5s on average (max 12s). Averages are computed w.r.t. non-timeout instances. The total number of executed clients is around 100. Only one TPC-C history was detected as inconsistent, which shows that the naive approach does not timeout only in the worst-case (inconsistency is a worst-case because all extensions and commit orders must be proved to be invalid).

A similar analysis on the TPC-C PC benchmark is omitted: TPC-C PC is a conflict-free variation of TPC-C with more operations per transaction. Thus, the rate of timeouts in the naive approach increases w.r.t. TPC-C, while the rate of timeouts using Algorithm 3 decreases.

Comparisons with prior work [7,4,18,20] are not possible as they do not apply to SQL (see Section 7 for more details).

This evaluation demonstrates that our algorithm scales well to practical testing workloads and that it outperforms brute-force search.

7 Related work

The formalization of database isolation levels has been considered in previous work. Adya [2] has proposed axiomatic specifications for isolation levels, which however do not concern more modern isolation levels like `PC` or `SI` and which are based on low-level modeling of database snapshots. We follow the more modern approach in [11,7] which however addresses the restricted case when transactions are formed of reads and writes on a *static* set of keys (variables) and not generic SQL queries, and all the transactions in a given execution have the same isolation level. Our axiomatic model builds on axioms defined by Biswas et al. [7] which are however applied on a new model of executions that is specific to SQL queries.

The complexity of checking consistency w.r.t isolation levels has been studied in [21,7]. The work of Papadimitriou [21] shows that checking serializability is NP-complete while the work of Biswas et al. [7] provides results for the same isolation levels as in our work, but in the restricted case mentioned above.

Checking consistency in a non-transactional case, shared-memory or distributed systems, has been investigated in a number of works, e.g., [9,16,13,10,17,14,1,15,3]. Transactions introduce additional challenges that make these results not applicable.

Existing tools for checking consistency in the transactional case of distributed databases, e.g., [7,4,18,20] cannot handle SQL-like semantics, offering guarantees modulo their transformations to reads and writes on static sets of keys. Our results show that handling the SQL-like semantics is strictly more complex (NP-hard in most cases).

Acknowledgements

We thank the anonymous reviewers for their feedback. This work was partially supported by the Agence National de Recherche (ANR) grant CENTEANES

References

1. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Bengt Jonsson, and Tuan Phong Ngo. Optimal stateless model checking under the release-acquire semantics. *Proc. ACM Program. Lang.*, 2(OOPSLA):135:1–135:29, 2018. doi:10.1145/3276505.
2. A. Adya. Weak consistency: A generalized theory and optimistic implementations for distributed transactions. Technical report, USA, 1999.
3. Pratyush Agarwal, Krishnendu Chatterjee, Shreya Pathak, Andreas Pavlogiannis, and Viktor Toman. Stateless model checking under a reads-value-from equivalence. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I*, volume 12759 of *Lecture Notes in Computer Science*, pages 341–366. Springer, 2021. doi:10.1007/978-3-030-81685-8_16.
4. Peter Alvaro and Kyle Kingsbury. Elle: Inferring isolation anomalies from experimental observations. *Proc. VLDB Endow.*, 14(3):268–280, 2020. URL: http://www.vldb.org/pvldb/vol14/p268-alvaro.pdf, doi:10.5555/3430915.3442427.
5. anonymous authors. Benchbase-evaluation, October 2024. URL: omittedforanonymity.
6. Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, and Patrick E. O'Neil. A critique of ANSI SQL isolation levels. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA, May 22-25, 1995*, pages 1–10. ACM Press, 1995. doi:10.1145/223784.223785.
7. Ranadeep Biswas and Constantin Enea. On the complexity of checking transactional consistency. *Proc. ACM Program. Lang.*, 3(OOPSLA):165:1–165:28, 2019. doi:10.1145/3360591.
8. Ranadeep Biswas, Diptanshu Kakwani, Jyothi Vedurada, Constantin Enea, and Akash Lal. Monkeydb: effectively testing correctness under weak isolation levels. *Proc. ACM Program. Lang.*, 5(OOPSLA):1–27, 2021. doi:10.1145/3485546.
9. Ahmed Bouajjani, Constantin Enea, Rachid Guerraoui, and Jad Hamza. On verifying causal consistency. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 626–638. ACM, 2017. doi:10.1145/3009837.3009888.
10. Jason F. Cantin, Mikko H. Lipasti, and James E. Smith. The complexity of verifying memory coherence and consistency. *IEEE Trans. Parallel Distributed Syst.*, 16(7):663–671, 2005. doi:10.1109/TPDS.2005.86.
11. Andrea Cerone, Giovanni Bernardi, and Alexey Gotsman. A framework for transactional consistency models with atomic visibility. In Luca Aceto and David de Frutos-Escrig, editors, *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, volume 42 of *LIPIcs*, pages 58–71. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. URL: https://doi.org/10.4230/LIPIcs.CONCUR.2015.58, doi:10.4230/LIPICS.CONCUR.2015.58.
12. Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. Oltp-bench: An extensible testbed for benchmarking relational databases. *Proc. VLDB Endow.*, 7(4):277–288, 2013. URL: http://www.vldb.org/pvldb/vol7/p277-difallah.pdf, doi:10.14778/2732240.2732246.
13. Michael Emmi and Constantin Enea. Sound, complete, and tractable linearizability monitoring for concurrent collections. *Proc. ACM Program. Lang.*, 2(POPL):25:1–25:27, 2018. doi:10.1145/3158113.

14. Florian Furbach, Roland Meyer, Klaus Schneider, and Maximilian Senftleben. Memory-model-aware testing: A unified complexity analysis. *ACM Trans. Embed. Comput. Syst.*, 14(4):63:1–63:25, 2015. doi:10.1145/2753761.
15. Phillip B. Gibbons and Ephraim Korach. On testing cache-coherent shared memories. In Lawrence Snyder and Charles E. Leiserson, editors, *Proceedings of the 6th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '94, Cape May, New Jersey, USA, June 27-29, 1994*, pages 177–188. ACM, 1994. doi:10.1145/181014.181328.
16. Phillip B. Gibbons and Ephraim Korach. Testing shared memories. *SIAM J. Comput.*, 26(4):1208–1244, 1997. doi:10.1137/S0097539794279614.
17. Alex Gontmakher, Sergey V. Polyakov, and Assaf Schuster. Complexity of verifying java shared memory execution. *Parallel Process. Lett.*, 13(4):721–733, 2003. doi:10.1142/S0129626403001628.
18. Kaile Huang, Si Liu, Zhenge Chen, Hengfeng Wei, David A. Basin, Haixiang Li, and Anqun Pan. Efficient black-box checking of snapshot isolation in databases. *Proc. VLDB Endow.*, 16(6):1264–1276, 2023. URL: https://www.vldb.org/pvldb/vol16/p1264-wei.pdf, doi:10.14778/3583140.3583145.
19. Jepsen. Distributed systems testing, 2020. https://jepsen.io/.
20. Si Liu, Long Gu, Hengfeng Wei, and David A. Basin. Plume: Efficient and complete black-box checking of weak isolation levels. *Proc. ACM Program. Lang.*, 8(OOPSLA2):876–904, 2024. doi:10.1145/3689742.
21. Christos H. Papadimitriou. The serializability of concurrent database updates. *J. ACM*, 26(4):631–653, 1979. doi:10.1145/322154.322158.
22. Andrew Pavlo. What are we doing with our lives?: Nobody cares about our concurrency control research. In Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, page 3. ACM, 2017. doi:10.1145/3035918.3056096.
23. Douglas B. Terry, Alan J. Demers, Karin Petersen, Mike Spreitzer, Marvin Theimer, and Brent B. Welch. Session guarantees for weakly consistent replicated data. In *Proceedings of the Third International Conference on Parallel and Distributed Information Systems (PDIS 94), Austin, Texas, USA, September 28-30, 1994*, pages 140–149. IEEE Computer Society, 1994. doi:10.1109/PDIS.1994.331722.
24. TPC. Technical report, Transaction Processing Performance Council, February 2010. URL: http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf.

A An operational semantics for SQL-like distributed databases (Section 3.3).

BEGIN

$$\frac{\begin{array}{c} t \text{ fresh} \quad e \text{ fresh} \quad \mathsf{P}(j) = \texttt{begin}(\iota); \mathsf{Body}; \texttt{commit}; \mathsf{P} \quad \mathbf{B}(j) = \epsilon \\ \tau = 1 + \max\{\mathbf{T}(e') \mid e' \in \mathsf{events}(h)\} \quad \mathbf{T}' = \mathbf{T}[e \to \tau] \quad \delta = \mathsf{snapshot}_\iota(h, \mathbf{S}, \mathbf{T}', e, \texttt{begin}) \\ h' = h \oplus_j (t, \iota, \{(e, \texttt{begin})\}, \emptyset) \end{array}}{h, \boldsymbol{\gamma}, \mathbf{B}, \mathbf{I}, \mathbf{T}, \mathbf{S}, \mathsf{P} \Rightarrow h', \boldsymbol{\gamma}[j \mapsto \emptyset], \mathbf{B}[j \mapsto \mathsf{Body}; \texttt{commit}], \mathbf{I}[t \mapsto \iota], \mathbf{T}', \mathbf{S}[e \mapsto \delta], \mathsf{P}[j \mapsto \mathsf{S}]}$$

IF-TRUE

$$\frac{\psi(\boldsymbol{a})[\boldsymbol{\gamma}(j)(a)/a : a \in \boldsymbol{a}] \quad \mathbf{B}(j) = \texttt{if}(\psi(\boldsymbol{a}))\{\mathsf{Instr}\}; \mathsf{B}}{h, \boldsymbol{\gamma}, \mathbf{B}, \mathbf{I}, \mathbf{T}, \mathbf{S}, \mathsf{P} \Rightarrow h, \boldsymbol{\gamma}, \mathbf{B}[j \mapsto \mathsf{Instr}; \mathsf{B}], \mathbf{I}, \mathbf{T}, \mathbf{S}, \mathsf{P}}$$

IF-FALSE

$$\frac{\neg\psi(\boldsymbol{a})[\boldsymbol{\gamma}(j)(a)/a : a \in \boldsymbol{a}] \quad \mathbf{B}(j) = \texttt{if}(\psi(\boldsymbol{x}))\{\mathsf{Instr}\}; \mathsf{B}}{h, \boldsymbol{\gamma}, \mathbf{B}, \mathbf{I}, \mathbf{T}, \mathbf{S}, \mathsf{P} \Rightarrow h, \boldsymbol{\gamma}, \mathbf{B}[j \mapsto \mathsf{B}], \mathbf{I}, \mathbf{T}, \mathbf{S}, \mathsf{P}}$$

LOCAL

$$\frac{v = e[\boldsymbol{\gamma}(j)(a')/a' : a' \in \boldsymbol{a'}] \quad \mathbf{B}(j) = a := e(\boldsymbol{a'}); \mathsf{B}}{h, \boldsymbol{\gamma}, \mathbf{B}, \mathbf{I}, \mathbf{T}, \mathbf{S}, \mathsf{P} \Rightarrow h, \boldsymbol{\gamma}[j, a \mapsto v], \mathbf{B}[j \mapsto \mathsf{B}], \mathbf{I}, \mathbf{T}, \mathbf{S}, \mathsf{P}}$$

COMMIT

$$\frac{\begin{array}{c} e \text{ fresh} \quad t = \texttt{last}(h, j) \quad \iota = \mathsf{iso}(h)(t) \quad \mathbf{B}(j) = \texttt{commit} \\ \tau = 1 + \max\{\mathbf{T}(e') \mid e' \in \mathsf{events}(h)\} \quad \mathbf{T}' = \mathbf{T}[e \to \tau] \\ \delta = \mathsf{snapshot}_\iota(h, \mathbf{S}, \mathbf{T}', e, \texttt{commit}) \quad \mathsf{validate}_\iota(h, \mathbf{T}', t) \end{array}}{h, \boldsymbol{\gamma}, \mathbf{B}, \mathbf{I}, \mathbf{T}, \mathsf{P} \Rightarrow h \oplus_j (e, \texttt{commit}), \boldsymbol{\gamma}, \mathbf{B}[j \mapsto \epsilon], \mathbf{I}, \mathbf{T}', \mathbf{S}[e \mapsto \delta], \mathsf{P}}$$

ABORT

$$\frac{\begin{array}{c} e \text{ fresh} \quad t = \texttt{last}(h, j) \quad \iota = \mathsf{iso}(h)(t) \quad \mathbf{B}(j) = \texttt{abort}; B \\ \tau = 1 + \max\{\mathbf{T}(e') \mid e' \in \mathsf{events}(h)\} \quad \mathbf{T}' = \mathbf{T}[e \to \tau] \\ \delta = \mathsf{snapshot}_\iota(h, \mathbf{S}, \mathbf{T}', e, \texttt{abort}) \quad \mathsf{validate}_\iota(h, \mathbf{T}', t) \end{array}}{h, \boldsymbol{\gamma}, \mathbf{B}, \mathbf{I}, \mathbf{T}, \mathbf{S}, \mathsf{P} \Rightarrow h \oplus_j (e, \texttt{abort}), \boldsymbol{\gamma}, \mathbf{B}[j \mapsto \epsilon], \mathbf{I}, \mathbf{T}', \mathbf{S}[e \mapsto \delta], \mathsf{P}}$$

Fig. A.1: An operational semantics for transactional programs. Above, $\texttt{last}(h, j)$ denotes the last transaction log in the session order $\mathsf{so}(j)$ of h while $\mathsf{snapshot}_\iota$ and $\mathsf{readFrom}$ denote the snapshot visible to an instruction and the writes it reads from, respectively. The $\mathsf{validate}_\iota$ checks if a transaction can be committed. They are defined in Figure A.3.

.

Formally, the operational semantics is defined as a transition relation $\Rightarrow$ between *configurations*. A configuration is a tuple containing the following:

- history h recording the instructions executed in the past,

$$\textsc{insert}\quad \frac{\begin{array}{c} e \text{ fresh} \quad t = \mathtt{last}(h, j) \quad \iota = \mathsf{iso}(h)(t) \quad \mathbf{B}(j) = \mathtt{INSERT}(\mathsf{R}); \mathsf{B} \\ \tau = 1 + \max\{\mathbf{T}(e') \mid e' \in \mathsf{events}(h)\} \quad \mathbf{T}' = \mathbf{T}[e \to \tau] \\ \delta = \mathsf{snapshot}_{\iota}(h, \mathbf{S}, \mathbf{T}', e, \mathtt{INSERT}) \quad h' = h \oplus_j (e, \mathtt{INSERT}(\mathsf{R})) \end{array}}{h, \boldsymbol{\gamma}, \mathbf{B}, \mathbf{I}, \mathbf{T}, \mathbf{S}, \mathsf{P} \Rightarrow h', \boldsymbol{\gamma}, \mathbf{B}[j \mapsto \mathsf{B}], \mathbf{I}, \mathbf{T}', \mathbf{S}[e \mapsto \delta], \mathsf{P}}$$

$$\textsc{select}\quad \frac{\begin{array}{c} e \text{ fresh} \quad t = \mathtt{last}(h, j) \quad \iota = \mathsf{iso}(h)(t) \quad \mathbf{B}(j) = a := \mathtt{SELECT}(\mathsf{p}); \mathsf{B} \\ \tau = 1 + \max\{\mathbf{T}(e') \mid e' \in \mathsf{events}(h)\} \quad \mathbf{T}' = \mathbf{T}[e \to \tau] \\ \delta = \mathsf{snapshot}_{\iota}(h, \mathbf{S}, \mathbf{T}', e, \mathtt{SELECT}) \quad \mathbf{w} = \mathsf{readFrom}(h, \mathbf{T}, t, \delta) \\ h' = (h \oplus_j (e, \mathtt{SELECT}(\mathsf{p}))) \bigoplus_{x \in \mathsf{Keys}, \mathbf{w}[x] \neq \bot} \mathsf{wr}(\mathbf{w}[x], e) \end{array}}{h, \boldsymbol{\gamma}, \mathbf{B}, \mathbf{I}, \mathbf{T}, \mathbf{S}, \mathsf{P} \Rightarrow h', \boldsymbol{\gamma}[(j, a) \mapsto \{\mathsf{r} \in \delta : \mathsf{p}(\mathsf{r})\}], \mathbf{B}[j \mapsto \mathsf{B}], \mathbf{I}, \mathbf{T}', \mathbf{S}[e \mapsto \delta], \mathsf{P}}$$

$$\textsc{update}\quad \frac{\begin{array}{c} e \text{ fresh} \quad t = \mathtt{last}(h, j) \quad \iota = \mathsf{iso}(h)(t) \quad \mathbf{B}(j) = \mathtt{UPDATE}(\mathsf{p},\ \mathsf{U}); \mathsf{B} \\ \tau = 1 + \max\{\mathbf{T}(e') \mid e' \in \mathsf{events}(h)\} \quad \mathbf{T}' = \mathbf{T}[e \to \tau] \\ \delta = \mathsf{snapshot}_{\iota}(h, \mathbf{S}, \mathbf{T}', e, \mathtt{UPDATE}) \quad \mathbf{w} = \mathsf{readFrom}(h, \mathbf{T}, t, \delta) \\ h' = (h \oplus_j (e, \mathtt{UPDATE}(\mathsf{p},\ \mathsf{U}))) \bigoplus_{x \in \mathsf{Keys}, \mathbf{w}[x] \neq \bot} \mathsf{wr}(\mathbf{w}[x], e) \end{array}}{h, \boldsymbol{\gamma}, \mathbf{B}, \mathbf{I}, \mathbf{T}, \mathbf{S}, \mathsf{P} \Rightarrow h', \boldsymbol{\gamma}, \mathbf{B}[j \mapsto \mathsf{B}], \mathbf{I}, \mathbf{T}', \mathbf{S}[e \mapsto \delta], \mathsf{P}}$$

$$\textsc{delete}\quad \frac{\begin{array}{c} e \text{ fresh} \quad t = \mathtt{last}(h, j) \quad \iota = \mathsf{iso}(h)(t) \quad \mathbf{B}(j) = \mathtt{DELETE}(\mathsf{p}); \mathsf{B} \\ \tau = 1 + \max\{\mathbf{T}(e') \mid e' \in \mathsf{events}(h)\} \quad \mathbf{T}' = \mathbf{T}[e \to \tau] \\ \delta = \mathsf{snapshot}_{\iota}(h, \mathbf{S}, \mathbf{T}', e, \mathtt{DELETE}) \quad \mathbf{w} = \mathsf{readFrom}(h, \mathbf{T}, t, \delta) \\ h' = (h \oplus_j (e, \mathtt{DELETE}(\mathsf{p}))) \bigoplus_{x \in \mathsf{Keys}, \mathbf{w}[x] \neq \bot} \mathsf{wr}(\mathbf{w}[x], e) \end{array}}{h, \boldsymbol{\gamma}, \mathbf{B}, \mathbf{I}, \mathbf{T}, \mathbf{S}, \mathsf{P} \Rightarrow h', \boldsymbol{\gamma}, \mathbf{B}[j \mapsto \mathsf{B}], \mathbf{I}, \mathbf{T}', \mathbf{S}[e \mapsto \delta], \mathsf{P}}$$

Fig. A.2: An operational semantics for transactional programs. Above, $\mathtt{last}(h, j)$ denotes the last transaction log in the session order $\mathsf{so}(j)$ of h while $\mathsf{snapshot}_{\iota}$ and $\mathsf{readFrom}$ denote the snapshot visible to an instruction and the writes it reads from, respectively. The $\mathsf{validate}_{\iota}$ checks if a transaction can be committed. They are defined in Figure A.3.

.

- a valuation map $\boldsymbol{\gamma}$ that records local variable values in the current transaction of each session ($\boldsymbol{\gamma}$ associates identifiers of sessions that have live transactions with valuations of local variables),
- a map $\mathbf{B}$ that stores the code of each live transaction (associating session identifiers with code),
- a map $\mathbf{I}$ that tracks the isolation level of each executed transaction,
- a map $\mathbf{T}$ that associates events in the history with unique timestamps,
- a map $\mathbf{S}$ that associates events in the history with snapshots of the database,
- sessions/transactions P that remain to be executed from the original program.

For readability, we define a program as a partial function $\mathsf{P} : \mathsf{SessId} \rightharpoonup \mathsf{Sess}$ that associates session identifiers in SessId with sequences of transactions as defined in Section 2.1. Similarly, the session order so in a history is defined as a partial function $\mathsf{so} : \mathsf{SessId} \rightharpoonup \mathsf{Tlogs}^*$ that associates session identifiers with sequences of transaction logs. Two transaction logs are ordered by so if one occurs before the other in some sequence $\mathsf{so}(j)$ with $j \in \mathsf{SessId}$.

Before presenting the definition of $\Rightarrow_I$, we introduce some notation. Let h be a history that contains a representation of so as above. We use $h \oplus_j (t, \iota_t, E, \mathsf{po}_t)$ to denote a history where $(t, \iota_t, E, \mathsf{po}_t)$ is appended to $\mathsf{so}(j)$. Also, for an event e, $h \oplus_j e$ is the history obtained from h by adding e to the last transaction log in $\mathsf{so}(j)$ and as a last event in the program order of this log (i.e., if $\mathsf{so}(j) = \sigma; (t, \iota_t, E, \mathsf{po}_t)$, then the session order so' of $h \oplus_j e$ is defined by $\mathsf{so}'(k) = \mathsf{so}(k)$ for all $k \neq j$ and $\mathsf{so}(j) = \sigma; (t, \iota_t, E \cup e, \mathsf{po} \cup \{(e', e) : e' \in E\})$). Finally, for a history $h = (T, \mathsf{so}, \mathsf{wr})$, $h \oplus \mathsf{wr}(t, e)$ is the history obtained from h by adding (t, e) to the write-read relation.

Figures A.1and A.2 list the rules defining $\Rightarrow$. We distinguish between local computation rules (IF-TRUE, IF-FALSE and LOCAL) and database-accesses rules (BEGIN, INSERT, SELECT, UPDATE, DELETE, COMMIT and ABORT); each associated to its homonymous instruction. Database-accesses get an increasing timestamp τ as well as an isolation-depending snapshot of the database using predicate $\mathsf{snapshot}_\iota$; updating adequately the timestamp and snapshot maps ($\mathbf{T}$ and $\mathbf{S}$ respectively). Timestamps are used for validating the writes of a transaction and blocking inconsistent runs as well as for defining the set of possible snapshots any event can get. We use predicate $\mathsf{readFrom}$ for determining the values read by an event. Those reads depend on both the event's snapshot as well as the timestamp of every previously executed event. Their formal definitions are described in Figure A.3.

The BEGIN rule starts a new transaction, provided that there is no other live transaction ($\mathsf{B} = \epsilon$) in the same session. It adds an empty transaction log to the history and schedules the body of the transaction. IF-TRUE and IF-FALSE check the truth value of a Boolean condition of an `if` conditional. LOCAL handles the case where some local computation is required. INSERT, SELECT, UPDATE and DELETE handle the database accesses. INSERT add some rows R in the history. SELECT, UPDATE and DELETE read every key from a combination of its snapshot and the local writes defined by $\mathsf{readFrom}$ function. The predicate $\_\ \mathsf{writes}\ \_$ implicitly uses the previous information stored in the history via the function $\mathsf{value}_{\mathsf{wr}}$. Finally COMMIT and ABORT validate that the run of the transaction correspond to the isolation level specification. These rules may block in case the validation is not satisfied as the predicate valuation does not change with the application of posterior rules.

An *initial* configuration for program P contains the program P along with a history $h = (\{t_0\}, \emptyset, \emptyset)$, where t_0 is a transaction log containing only writes that write the initial values of all keys and whose timestamp and snapshot is 0 ($\mathbf{S}, \mathbf{T} = [t_0 \mapsto 0]$), and it does not contain transaction code nor local keys ($\boldsymbol{\gamma}, \mathbf{B} = \emptyset$). A *run* ρ of a program P is a sequence of configurations $c_0 c_1 \ldots c_n$

where c_0 is an initial configuration for P, and $c_m \Rightarrow c_{m+1}$, for every $0 \leq m < n$. We say that c_n is *reachable* from c_0. The history of such a run, $\mathsf{history}(\rho)$, is the history h_n in the last configuration c_n. A configuration is called *final* if it contains the empty program ($\mathsf{P} = \emptyset$). Let $\mathsf{hist}(\mathsf{P})$ denote the set of all histories of a run of P that ends in a final configuration.

$$\mathsf{snapshot}_{\mathtt{SER}}(h, \mathbf{S}, \mathbf{T}', e, \xi) = \begin{cases} \max\left\{ \mathbf{T}'(c_{t'}) \;\middle|\; \begin{array}{l} t' \in h \wedge \\ c_{t'} = \mathtt{commit}(t') \wedge \\ \mathbf{T}'(c_{t'}) < \mathbf{T}'(e) \end{array} \right\} & \text{if } \xi = \mathtt{begin} \\ \mathbf{S}(\mathtt{begin}(\mathsf{tr}(e))) & \text{otherwise} \end{cases}$$

$$\mathsf{snapshot}_{\mathtt{SI}}(h, \mathbf{S}, \mathbf{T}', e, \xi) = \begin{cases} \mathsf{choice}\left(\left\{ \mathbf{T}'(c_{t'}) \;\middle|\; \begin{array}{l} t' \in h \wedge \\ c_{t'} = \mathtt{commit}(t') \wedge \\ vec\mathsf{T}'(c_{t'}) < \mathbf{T}'(e) \end{array} \right\}\right) & \text{if } \xi = \mathtt{begin} \\ \mathbf{S}(\mathtt{begin}(\mathsf{tr}(e))) & \text{otherwise} \end{cases}$$

$$\mathsf{snapshot}_{\mathtt{RC}}(h, \mathbf{S}, \mathbf{T}', e, \xi) = \mathsf{choice}\left(\left\{ \mathbf{T}'(c_{t'}) \;\middle|\; \begin{array}{l} t' \in h \wedge \\ c_{t'} = \mathtt{commit}(t') \wedge \mathbf{T}'(c_{t'}) < \mathbf{T}'(e) \wedge \\ \forall e'. \left(\begin{array}{l} (e', e) \in \mathsf{po} \vee \\ (\mathsf{tr}(e'), \mathsf{tr}(e)) \in \mathsf{so} \end{array}\right) \\ \qquad\qquad \implies \mathbf{S}(e') \leq \mathbf{T}(c_{t'}) \end{array} \right\}\right)$$

$$\mathsf{readFrom}(h, \mathbf{T}, t, \delta) = [x \mapsto (\mathtt{localWr}[x] \neq \bot)\,?\,\bot \;:\; w_x \text{ for each } x \in \mathsf{Keys}]$$

$$\text{where } \mathtt{localWr}[x] = \max_{\mathsf{po}}\{e \mid \mathsf{tr}(e) = t \;\wedge\; e \text{ writes } x\} \cup \{\bot\}$$

$$\text{and } w_x \text{ writes } x \wedge \mathbf{T}(w_x) = \max\left\{ \mathbf{T}(w') \;\middle|\; \begin{array}{c} w' \in \mathsf{events}(h) \;\wedge\; w' \text{ writes } x \;\wedge \\ \mathbf{T}(\mathtt{commit}(\mathsf{tr}(w'))) \leq \delta \end{array} \right\}$$

$$\mathsf{validate}_{\mathtt{SER}}(h, \mathbf{T}', t) = \left(\begin{array}{c} \nexists t' \in h, x \in \mathsf{Keys} \text{ s.t. } (t \text{ reads } x \vee t \text{ writes } x) \;\wedge\; t' \text{ writes } x \\ \wedge\; \mathbf{T}'(\mathtt{begin}(t)) < \mathbf{T}'(\mathtt{commit}(t')) < \mathbf{T}'(\mathtt{end}(t)) \end{array} \right)$$

$$\mathsf{validate}_{\mathtt{SI}}(h, \mathbf{T}', t) = \left(\begin{array}{c} \nexists t' \in h, x \in \mathsf{Keys} \text{ s.t. } t \text{ writes } x \;\wedge\; t' \text{ writes } x \;\wedge \\ \wedge\; \mathbf{T}'(\mathtt{begin}(t)) < \mathbf{T}'(\mathtt{commit}(t')) < \mathbf{T}'(\mathtt{end}(t)) \end{array} \right)$$

$$\mathsf{validate}_{\mathtt{RC}}(h, \mathbf{T}', t) = \;\mathsf{true}$$

Fig. A.3: Definition of auxiliary functions for the operational semantics. The function choice receives a set as input and returns one of its elements.

.

The proof of Theorem 1 is split in two parts: Lemma 2 and Lemma 4. In Lemma 2, we prove by induction that for any run ρ, $\mathsf{history}(\rho)$ is a full history; using the auxiliary Lemma 1 about pending transactions. We then define in

Equation 5 a relation on transactions that plays the role of consistency witness for $\mathsf{history}(\rho)$. Then, we prove in Lemma 3 that such relation is a commit order for $\mathsf{history}(\rho)$ to conclude in Lemma 4 that $\mathsf{history}(\rho)$ is indeed consistent. In all cases, we do a case-by-case analysis depending on which rule is employed during the inductive step.

For the sake of simplifying our notation, we denote by $\mathsf{rule}(\rho, j, \rho')$ to the rule s.t. applied to run ρ on session j leads to configuration ρ'.

Lemma 1. *Let ρ be a run and $\mathsf{history}(\rho) = (T, \mathsf{so}, \mathsf{wr})$ be its history. Any pending transaction in T is $(\mathsf{so} \cup \mathsf{wr})$-maximal.*

Proof. We prove by induction on the length of a run ρ that any pending transaction is $(\mathsf{so} \cup \mathsf{wr})$-maximal; where $\mathsf{history}(\rho) = (T, \mathsf{so}, \mathsf{wr})$. The base case, where $\rho = \{c_0\}$ and c_0 is an initial configuration, is immediate by definition. Let us suppose that for every run of length at most n the property holds and let ρ' a run of length $n + 1$. As ρ' is a sequence of configurations, there exist a reachable run ρ of length n, a session j and a rule r s.t. $r = \mathsf{rule}(\rho, j, \rho')$. Let us call $h = (T, \mathsf{so}, \mathsf{wr})$, $h' = (T', \mathsf{so}', \mathsf{wr}')$ and e to $\mathsf{history}(\rho)$, $\mathsf{history}(\rho')$ and the last event in po-order belonging to $\mathtt{last}(h, j)$ respectively. By induction hypothesis, any pending transaction in h is $(\mathsf{so} \cup \mathsf{wr})$-maximal. To conclude the inductive step, we show that for every possible rule r s.t. $r = \mathsf{rule}(\rho, j, \rho')$, the property also holds in h'.

- LOCAL, IF-FALSE, IF-TRUE, INSERT, COMMIT, ABORT: The result trivially holds as $\mathsf{wr}' = \mathsf{wr}$, $\mathsf{so}' = \mathsf{so}$ and $\mathsf{complete}(T') \subseteq \mathsf{complete}(T)$.
- BEGIN: We observe that in this case, $T = T \cup \{\mathtt{last}(h, j)\}$, $\mathsf{reads}(T') = \mathsf{reads}(T)$, $\mathsf{wr}' = \mathsf{wr}$ and $\mathsf{so}' = \mathsf{so} \cup \{(t', \mathtt{last}(h, j)) \mid \mathtt{ses}(t') = j\}$. Thus, $\mathtt{last}(h, j)$ is pending and $\mathsf{so}' \cup \mathsf{wr}'$-maximal. Moreover, as described in Figure A.1, $\mathbf{B}(j) = \epsilon$; so there is no other transaction in session j that is pending. Hence, as $T' \setminus \mathsf{complete}(T') = T\mathsf{complete}(T) \cup \{\mathtt{last}(h, j)\}$, by induction hypothesis, every pending transaction is $\mathsf{so}' \cup \mathsf{wr}'$-maximal.
- SELECT, UPDATE, DELETE: Figure A.2 describes h' by the equation $h' = (h \oplus_j (e, \mathsf{rule}(\rho, j, s))) \bigoplus_{x \in \mathsf{Keys}, \mathbf{w}[x] \neq \bot} \mathsf{wr}(\mathbf{w}[x], e)$; where e is the new event executed and $\mathbf{w}$ is defined following the descriptions in Figures A.2 and A.3. In this case, $T' = T, \mathsf{reads}(T') = \mathsf{reads}(T) \cup \{e\}$, $\mathsf{so}' = \mathsf{so}$, $\forall x \in \mathsf{Keys}$ s.t. $\mathbf{w}[x] = \bot$, $\mathsf{wr}'_x = \mathsf{wr}_x$ and $\forall x \in \mathsf{Keys}$ s.t. $\mathbf{w}[x] \neq \bot$, $\mathsf{wr}'_x = \mathsf{wr}_x \cup \{(\mathbf{w}[x], e)\}$. Note that as described by Figure A.3, in the latter case, when $\mathbf{w}[x] \neq \bot$, $\mathsf{tr}(\mathbf{w}[x]) \in \mathsf{complete}(T) = \mathsf{complete}(T')$. In conclusion, using the induction hypothesis, we also conclude that every pending transaction is $\mathsf{so}' \cup \mathsf{wr}'$-maximal.

□

Lemma 2. *For every run ρ, $\mathsf{history}(\rho)$ is a full history.*

Proof. We prove by induction on the length of a run ρ that $\mathsf{history}(\rho)$ is a full history; where the base case, $\rho = \{c_0\}$ and c_0 is an initial configuration, is trivial by definition. Let us suppose that for every run of length at most n the property holds and let ρ' a run of length $n+1$. As ρ' is a sequence of configurations, there exist a reachable run ρ of length n, a session j and a rule r s.t. $r = \mathsf{rule}(\rho, j, \rho')$. Let us call $h = (T, \mathsf{so}, \mathsf{wr})$, $h' = (T', \mathsf{so}', \mathsf{wr}')$ and e to $\mathsf{history}(\rho)$, $\mathsf{history}(\rho')$ and the last event in po-order belonging to $\mathtt{last}(h, j)$ respectively. By induction hypothesis, h is a full history. To conclude the inductive step, we show that for every possible rule r s.t. $r = \mathsf{rule}(\rho, j, \rho')$, the history h' is also a full history. In particular, by definitions 1 and 2, it suffices to prove that $\mathsf{so}' \cup \mathsf{wr}'$ is an acyclic relation and that for every variable x and read event r, ${\mathsf{wr}'_x}^{-1}(r)\downarrow$ if and only if r does not read x from a local write and in such case, $\mathtt{value}_{\mathsf{wr}'}({\mathsf{wr}'_x}^{-1}(r), x) \neq \bot$.

- LOCAL, IF-FALSE, IF-TRUE, INSERT, COMMIT, ABORT: The result trivially holds as $\mathsf{reads}(T') = \mathsf{reads}(T), \mathsf{wr}' = \mathsf{wr}$ and $\mathsf{so}' = \mathsf{so}$; using that h is consistent.
- BEGIN: We observe that $h' = h \oplus_j (e, \mathtt{begin})$, so $T = T \cup \{\mathtt{last}(h, j)\}$, $\mathsf{reads}(T') = \mathsf{reads}(T)$, $\mathsf{wr}' = \mathsf{wr}$ and $\mathsf{so}' = \mathsf{so} \cup \{(t', \mathtt{last}(h, j)) \mid \mathtt{ses}(t') = j\}$. In such case, by Lemma 1, t is $\mathsf{so}' \cup \mathsf{wr}'$-maximal. Thus, $\mathsf{so}' \cup \mathsf{wr}'$ is acyclic as $\mathsf{so} \cup \mathsf{wr}$ is also acyclic. Finally, as $\mathsf{wr}' = \mathsf{wr}$, we conclude that h' is a full history.
- SELECT, UPDATE, DELETE: Here $h' = (h \oplus_j (e, \mathsf{rule}(\rho, j, s))) \bigoplus_{x \in \mathsf{Keys}, \mathbf{w}[x] \neq \bot} \mathsf{wr}(\mathbf{w}[x], e)$ where e is the new event executed and $\mathbf{w}$ is defined following the descriptions in Figures A.2 and A.3. In this case, $T' = T, \mathsf{reads}(T') = \mathsf{reads}(T) \cup \{e\}$, $\mathsf{so}' = \mathsf{so}$, $\forall x \in \mathsf{Keys}$ s.t. $\mathbf{w}[x] = \bot$, $\mathsf{wr}'_x = \mathsf{wr}_x$ and $\forall x \in \mathsf{Keys}$ s.t. $\mathbf{w}[x] \neq \bot$, $\mathsf{wr}'_x = \mathsf{wr}_x \cup \{(\mathbf{w}[x], e)\}$. Note that as the timestamp of any event is always positive and $\mathbf{T}(\mathtt{init}) = 0$; for any key x, $\mathsf{w}[x] \neq \bot$ if and only if $\mathtt{localWr}[x] = \bot$. Thus, $\mathbf{w}$ is well defined, and ${\mathsf{wr}'_x}^{-1}(r)\downarrow$ if and only $\mathtt{localWr}[x] = \bot$. In such case, as any event w writes on a key x if and only if $\mathtt{value}_{\mathsf{wr}}(w, x) \neq \bot$, we conclude that $\mathtt{value}_{\mathsf{wr}'}({\mathsf{wr}'_x}^{-1}(r), x) \neq \bot$. To conclude the result, we need to show that $\mathsf{so}' \cup \mathsf{wr}'$ is acyclic. As ρ is reachable, by Figure A.3's definition we know that for any event r and key x, if $\mathsf{wr}_x^{-1}(r)\downarrow$, $\mathsf{tr}(\mathsf{wr}_x^{-1}(r)) \in \mathsf{cmtt}(h)$. Thus, by Lemma 1, $\mathtt{last}(h, j)$ is $\mathsf{so}' \cup \mathsf{wr}'$-maximal as it is not committed. Therefore, by the definition of so' and wr', as $\mathsf{so} \cup \mathsf{wr}$ is acyclic and $\mathtt{last}(h, j)$ is $\mathsf{so}' \cup \mathsf{wr}'$-maximal, $\mathsf{so}' \cup \mathsf{wr}'$ is also acyclic. In conclusion, h' is a full history.

□

Once proven that for any run ρ, $\mathsf{history}(\rho)$ is a full history, we need to prove that there exists a commit order co_ρ that witnesses $\mathsf{history}(\rho)$ consistency. Equation 5 defines a relation that we prove in Lemma 3 that it is a total order for $\mathsf{history}(\rho)$.

$$(t, t') \in \mathsf{co}_\rho \iff \begin{cases} t \in \mathsf{complete}(T) \land t' \in \mathsf{complete}(T) \land \\ \qquad \mathbf{T}(\mathtt{end}(t)) < \mathbf{T}(\mathtt{end}(t')) \\ t \in \mathsf{complete}(T) \land t' \notin \mathsf{complete}(T) \\ t \notin \mathsf{complete}(T) \land t' \notin \mathsf{complete}(T) \land \\ \qquad \mathbf{T}(\mathtt{begin}(t)) < \mathbf{T}(\mathtt{begin}(t')) \end{cases} \tag{5}$$

Lemma 3. *For every run ρ, the relation co_ρ defined above is a commit order for $\mathsf{history}(\rho)$.*

Proof. We prove by induction on the length of a run ρ that the relation co_ρ defined by the equation below is a commit order for $\mathsf{history}(\rho)$, i.e., if $\mathsf{history}(\rho) = (T, \mathsf{so}, \mathsf{wr})$, then $\mathsf{so} \cup \mathsf{wr} \subseteq \mathsf{co}_\rho$.

The base case, where ρ is composed only by an initial configuration is immediate as in such case $\mathsf{wr} = \emptyset$. Let us suppose that for every run of length at most n the property holds and let ρ' a run of length $n+1$. As ρ' is a sequence of configurations, there exist a reachable run ρ of length n, a session j and a rule r s.t. $\mathsf{r} = \mathsf{rule}(\rho, j, \rho')$. Let us call $h = (T, \mathsf{so}, \mathsf{wr})$, $h' = (T', \mathsf{so}', \mathsf{wr}')$ and e to $\mathsf{history}(\rho)$, $\mathsf{history}(\rho')$ and the last event in po-order belonging to $\mathtt{last}(h, j)$ respectively. By induction hypothesis, co_ρ is a commit order for h. To conclude the inductive step, we show that $\mathsf{co}_{\rho'}$ is also a commit order for h'.

- LOCAL, IF-FALSE, IF-TRUE: As $h = h'$ and $\mathbf{T}_{\rho'} = \mathbf{T}_\rho$, $\mathsf{co}_{\rho'} = \mathsf{co}_\rho$. Thus, the result trivially holds.
- BEGIN: In this case, $e = \mathtt{begin}(\mathtt{last}(h, j))$ and $\mathtt{last}(h, j) \notin \mathsf{complete}(T_{\rho'})$. Note that for any event $e' \neq e$, $\mathbf{T}(e) > \mathbf{T}(e')$ and $\mathsf{complete}(T'_\rho) = \mathsf{complete}(\rho)$. Thus, $\mathsf{co}_{\rho'} = \mathsf{co}_\rho \cup \{(t', \mathtt{last}(h', j)) \mid t' \in T\}$. As $\mathsf{so} \cup \mathsf{wr} \subseteq \mathsf{co}_\rho$, $\mathsf{wr}' = \mathsf{wr}$ and $\mathsf{so}' = \mathsf{so} \cup \{(t', \mathtt{last}(h, j)) \mid \mathtt{ses}(t') = j\}$, $\mathsf{so}' \cup \mathsf{wr}' \subseteq \mathsf{co}_{\rho'}$; so $\mathsf{co}_{\rho'}$ is a commit order for h'.
- INSERT: In this case, as $\mathsf{complete}(T'_\rho) = \mathsf{complete}(T_\rho)$, $\mathsf{co}_{\rho'} = \mathsf{co}_\rho$. Hence, as $\mathsf{so}' = \mathsf{so}$ and $\mathsf{wr}' = \mathsf{wr}$, $\mathsf{so}' \cup \mathsf{wr}' \subseteq \mathsf{co}_{\rho'}$.
- SELECT, UPDATE, DELETE: Once again, as $\mathsf{complete}(T'_\rho) = \mathsf{complete}(T_\rho)$, $\mathsf{co}_{\rho'} = \mathsf{co}_\rho$. Note that $\mathsf{so}' = \mathsf{so}$, $\forall x \in \mathsf{Keys}$ s.t. $\mathbf{w}[x] = \bot$, $\mathsf{wr}'_x = \mathsf{wr}_x$ and $\forall x \in \mathsf{Keys}$ s.t. $\mathbf{w}[x] \neq \bot$, $\mathsf{wr}'_x = \mathsf{wr}_x \cup \{(\mathbf{w}[x], e)\}$. In the latter case, where $\mathbf{w}[x] \neq \bot$, we know that $\mathsf{tr}(\mathbf{w}[x]) \in \mathsf{complete}(T)$ thanks to the definitions on Figure A.3. By Equation 5, as $\mathtt{last}(h, j)$ is pending, we deduce that $(\mathsf{tr}(\mathbf{w}[x]), \mathsf{tr}(e)) \in \mathsf{co}_{\rho'}$. Therefore, as $\mathsf{so} \cup \mathsf{wr} \subseteq \mathsf{co}_\rho = \mathsf{co}_{\rho'}$, we conclude that $\mathsf{so}' \cup \mathsf{wr}' \subseteq \mathsf{co}_{\rho'}$.
- COMMIT, ABORT: In this case, $e = \mathtt{endlast}(h, j)$, $\mathsf{co}_{\rho'} \restriction_{T \setminus \{\mathtt{last}(h,j)\} \times T \setminus \{\mathtt{last}(h,j)\}} = \mathsf{co}_\rho \restriction_{T \setminus \{\mathtt{last}(h,j)\} \times T \setminus \{\mathtt{last}(h,j)\}}$, $\mathsf{so}' = \mathsf{so}$ and $\mathsf{wr}' = \mathsf{wr}$. Thus, to prove that $\mathsf{so}' \cup \mathsf{wr}' \subseteq \mathsf{co}_{\rho'}$ we only need to discuss about $\mathtt{last}(h, j)$. By Lemma 1, $\mathtt{last}(h, j)$ is $\mathsf{so}' \cup \mathsf{wr}'$-maximal. Hence, we focus on proving that for any transaction t' s.t. $(t', \mathtt{last}(h, j)) \in \mathsf{so}' \cup \mathsf{wr}'$, $(t', \mathtt{last}(h, j)) \in \mathsf{co}_{\rho'}$. Any such transaction t' must be completed by Lemma 1. However, by the definition on Figure A.1, we know that $\mathbf{T}(e) > \mathbf{T}(\mathtt{end}(t'))$, so $(t', \mathtt{last}(h, j)) \in \mathsf{co}_{\rho'}$ by Equation 5. Thus, $\mathsf{so}' \cup \mathsf{wr}' \subseteq \mathsf{co}_{\rho'}$.

□

Lemma 4. *For every total run ρ, the $\mathsf{history}(\rho)$ is consistent.*

Proof. Let ρ^T be a total run. By Lemma 2, $\mathsf{history}(\rho^T)$ is a full history. Thus, to prove that $\mathsf{history}(\rho)$ is consistent, by Definition 6, we need to show that there exists a commit order co that witnesses its consistency. We prove by induction on the length of a prefix ρ of a total run ρ^T that the relation co_ρ defined in Equation 5 is a commit order that witnesses $\mathsf{history}(\rho)$'s consistency. Note that by Lemma 3, the relation co_ρ is indeed a commit order.

The base case, where ρ is composed only by an initial configuration is immediate as in such case $\mathsf{wr} = \emptyset$. Let us suppose that for every run of length at most n the property holds and let ρ' a run of length $n+1$. As ρ' is a sequence of configurations, there exist a reachable run ρ of length n, a session j and a rule r s.t. $\mathsf{r} = \mathsf{rule}(\rho, j, \rho')$. Let us call $h = (T, \mathsf{so}, \mathsf{wr})$, $h' = (T', \mathsf{so}', \mathsf{wr}')$ and e to $\mathsf{history}(\rho)$, $\mathsf{history}(\rho')$ and the last event in po-order belonging to $\mathtt{last}(h, j)$ respectively. By induction hypothesis, co_ρ is a commit order that witnesses h's consistency. To conclude the inductive step, we show that for every possible rule r s.t. $\mathsf{r} = \mathsf{rule}(\rho, j, \rho')$, $\mathsf{co}_{\rho'}$ is a commit order witnessing h''s consistency.

By contradiction, let suppose that $\mathsf{co}_{\rho'}$ does not witness h''s consistency. Then, there exists a variable x, a read event r, an axiom $a \in \iota$ and two committed transactions t_1, t_2 s.t. $(t_1, e) \in \mathsf{wr}_x$, t_2 writes x, $\mathsf{vis}_a^{\mathsf{co}_{\rho'}}(t_2, r, x)$ holds in h' but $(t_1, t_2) \in \mathsf{co}_{\rho'}$; where $\iota = \mathbf{I}(\mathtt{begin}(e))$. Thus, if we prove that such dependencies can be seen in h using co_ρ, we obtain a contradiction as co_ρ witnesses h's consistency. Note that as shown during the proof of Lemma 3, $\mathsf{co}_{\rho'} \restriction_{T \setminus \{\mathtt{last}(h,j)\} \times T \setminus \{\mathtt{last}(h,j)\}} = \mathsf{co}_\rho \restriction_{T \setminus \{\mathtt{last}(h,j)\} \times T \setminus \{\mathtt{last}(h,j)\}}$; so we simply prove that $\mathtt{last}(h, j)$ cannot be t_1, t_2, $\mathsf{tr}(r)$ or any intermediate transaction causing $\mathsf{vis}_a^{\mathsf{co}_{\rho'}}(t_2, r, x)$ to hold in h'.

- LOCAL, IF-FALSE, IF-TRUE: As $h = h'$ and $\mathsf{co}_{\rho'} = \mathsf{co}_\rho$, this case is impossible.
- BEGIN: In this case, $\mathsf{co}_{\rho'} = \mathsf{co}_\rho \cup \{(t', \mathtt{last}(h', j)) \mid t' \in T\}$. By Lemma 3, $\mathtt{last}(h', j)$ is $(\mathsf{so}' \cup \mathsf{wr}')$-maximal, so $\mathtt{last}(h', j) \neq t_1$. Moreover, $\mathsf{reads}(\mathtt{last}(h', j)) = \emptyset$, so $r \neq \mathsf{reads}(\mathtt{last}(h', j))$. In addition, $\mathtt{last}(h, j) \neq t_2$ as $\mathsf{writes}(\mathtt{last}(h, j)) = \emptyset$.
  - $\underline{a = \mathsf{Serializability}, \mathsf{Prefix} \text{ or } \mathsf{Read\ Committed}}$: In all cases, the axioms do not relate any other transactions besides t_1, t_2 and $\mathsf{tr}(r)$, so this case is impossible.
  - $\underline{a = \mathsf{Conflict}}$: In this case, $\mathtt{last}(h, j) \neq t_4$ as it is $\mathsf{co}_{\rho'}$-maximal; so this case is also impossible.
- INSERT: In this case, $\mathsf{co}_{\rho'} = \mathsf{co}_\rho$. Moreover, $\mathsf{reads}(T') = \mathsf{reads}(T)$, $\mathsf{writes}(T') = \mathsf{writes}(T)$, $\mathsf{so}' = \mathsf{so}$ and $\mathsf{wr}' = \mathsf{wr}$. Thus, this case is also impossible.
- SELECT, UPDATE, DELETE: In this case, $\mathsf{co}_{\rho'} = \mathsf{co}_\rho$, $\mathsf{so}' = \mathsf{so}$, $\forall x \in \mathsf{Keys}$ s.t. $\mathbf{w}[x] = \bot$, $\mathsf{wr}'_x = \mathsf{wr}_x$ and $\forall x \in \mathsf{Keys}$ s.t. $\mathbf{w}[x] \neq \bot$, $\mathsf{wr}'_x = \mathsf{wr}_x \cup \{(\mathbf{w}[x], e)\}$. As $\mathtt{last}(h, j)$ is pending, by Lemma 3, $\mathtt{last}(h, j) \neq t_1$ as it is $(\mathsf{so}' \cup \mathsf{wr}')$-maximal. Moreover, as $\mathsf{writes}(\mathtt{last}(h, j)) = \emptyset$, $\mathtt{last}(h, j) \neq t_2$. Then, we analyze if $\mathtt{last}(h, j)$ can be $\mathsf{tr}(r)$ (and thus, $r = e$) or any intermediate transaction. Note that for all three isolation levels we study, $\mathsf{readFrom}$ returns the value written by the transaction with the last commit timestamp for a given snapshot time. Hence, as $(t_1, r) \in \mathsf{wr}_x$ and $(t_2, \mathsf{tr}(r)) \in \mathsf{co}_\rho$, we deduce that $\mathbf{T}_\rho(\mathtt{commit}(t_2)) > \mathbf{T}_\rho(\mathtt{begin}(\mathtt{last}(h, j)))$. We continue the analysis distinguishing between one case per axiom:

- $\underline{a = \mathsf{Serializability}}$: As ρ' is a prefix of a total run ρ^T, there exists runs $\hat{\rho}, \hat{\rho}'$ s.t. $\mathsf{rule}(\hat{\rho}, j', \hat{\rho}')$ is either COMMIT or ABORT and both a prefix of ρ^T; where j' is the session of $\mathsf{tr}(r)$. Without loss of generality, we can assume that $\hat{\rho}$ and $\hat{\rho}'$ have minimal size; so $\mathtt{last}(\mathsf{history}(\hat{\rho}), j') = \mathsf{tr}(r)$. As ρ^T is total and $\hat{\rho}'$ is a prefix of ρ^T, $\mathsf{validate}_\iota(\mathsf{history}(\hat{\rho}, \mathbf{T}_{\hat{\rho}'}, \mathsf{tr}(r)))$ holds.
  By the monotonicity of $\mathbf{T}$, $\mathbf{T}_{\rho'} \subseteq \mathbf{T}_{\hat{\rho}'}$. Hence, as $(t_1, r) \in \mathsf{wr}_x$ and $\mathbf{T}_{\hat{\rho}'}(\mathtt{commit}(t_1)) < \mathbf{T}_{\hat{\rho}'}(\mathtt{commit}(t_2))$, by the definitions of Figure A.2 and Figure A.3 we deduce that $\mathbf{T}_{\hat{\rho}'}(\mathtt{begin}(\mathsf{tr}(r))) < \mathbf{T}_{\hat{\rho}'}(\mathtt{commit}(t_2))$. However, as $\mathbf{T}_{\hat{\rho}'}(\mathtt{begin}(\mathsf{tr}(r))) < \mathbf{T}_{\hat{\rho}'}(\mathtt{commit}(t_2)) < \mathbf{T}_{\hat{\rho}'}(\mathsf{end}(\mathsf{tr}(r)))$, $\mathsf{tr}(r)$ reads x, t_2 writes x; we conclude that $\mathsf{validate}_{\mathtt{SER}}(\mathsf{history}(\hat{\rho}'), \mathbf{T}_{\hat{\rho}'}, \mathsf{tr}(r))$ does not hold; so this case is impossible.
- $\underline{a = \mathsf{Conflict}}$: In this case, $\mathtt{last}(h, j)$ cannot be an intermediate transaction nor $\mathsf{tr}(r)$ as $\mathsf{writes}(\mathtt{last}(h, j)) = \emptyset$; so this case is also impossible.
- $\underline{a = \mathsf{Prefix}}$: In this case, $\mathtt{last}(h, j)$ cannot be an intermediate transaction as by Lemma 1, $\mathtt{last}(h, j)$ is $\mathsf{so}' \cup \mathsf{wr}'$-maximal. Thus, $\mathtt{last}(h, j)$ must be $\mathsf{tr}(r)$ and $e = r$. Therefore, there exists a transaction t_4 s.t. $(t_2, t_4) \in \mathsf{co}_{\rho'}{}^*$ and $(t_4, \mathtt{last}(h, j)) \in (\mathsf{so}' \cup \mathsf{wr}')$. Note that t_4 must be committed and that $\mathbf{T}_{\rho'}(\mathtt{commit}(t_4)) < \mathbf{T}_{\rho'}(\mathtt{begin}(\mathtt{last}(h, j)))$. Hence, as $(t_2, t_4) \in \mathsf{co}_{\rho'}{}^*$ and $(t_1, t_2) \in \mathsf{co}_{\rho'}$ and they are both committed, we deduce that $\mathbf{T}_{\rho'}(\mathtt{commit}(t_2)) < \mathbf{T}_{\rho'}(\mathtt{commit}(t_4)) < \mathbf{T}_{\rho'}(\mathtt{begin}(\mathtt{last}(h, j)))$. However, this contradicts that $\mathbf{T}_{\rho'}(\mathtt{commit}(t_2)) > \mathbf{T}_{\rho'}(\mathtt{begin}(\mathtt{last}(h, j)))$ Thus, this case is impossible.
- $\underline{a = \mathsf{Read\ Committed}}$: In this case, $\mathtt{last}(h, j)$ must be $\mathsf{tr}(r)$ and in particular, $e = r$. As depicted on Figure A.2 and Figure A.3, as $(t_1, r) \in \mathsf{wr}_x$, $\mathbf{S}_{\rho'}(e) \leq \mathbf{T}_{\rho'}(t_1)$. However, as $(t_1, t_2) \in \mathsf{co}_{\rho'}$, $\mathbf{T}_{\rho'}(\mathtt{commit}(t_1)) < \mathbf{T}_{\rho'}(\mathtt{commit}(t_2))$. Hence, as $(t_2, e) \in (\mathsf{so} \cup \mathsf{wr}); \mathsf{po}^*$, there exists an event $e' \in \mathtt{last}(h, j)$ s.t. $(e, e') \in \mathsf{po}^*$ and $\mathbf{T}_{\rho'}(\mathtt{commit}(t_2)) < \mathbf{T}_{\rho'}(e')$. However, by $\mathsf{snapshot}_{\mathtt{RC}}$'s definition, $\mathbf{S}(e') \leq \mathbf{S}_{\rho'}(e)$; so we deduce that $\mathbf{T}_{\rho'}(\mathtt{commit}(t_1)) < \mathbf{T}_{\rho'}(\mathtt{commit}(t_2)) < \mathbf{S}_{\rho'}(e)$. This contradicts the definition of $\mathsf{readFrom}$; so this case is impossible.

- COMMIT, ABORT: In this case, $\mathsf{co}_{\rho'} \restriction (T \setminus \{\mathtt{last}(h, j)\} \times T \setminus \{\mathtt{last}(h, j)\}) = \mathsf{co}_\rho \restriction (T \setminus \{\mathtt{last}(h, j)\} \times T \setminus \{\mathtt{last}(h, j)\})$, $\mathsf{so}' = \mathsf{so}$, $\mathsf{wr}' = \mathsf{wr}$. First, using that by induction hypothesis any prefix $\tilde{\rho}$ of ρ is consistent using $\mathsf{co}_{\tilde{\rho}}$; we define $\tilde{\rho}$ the prefix of ρ that introduces the read event r. As $\mathsf{history}(\tilde{\rho}) = (\tilde{T}, \tilde{\mathsf{so}}, \tilde{\mathsf{wr}})$ is consistent and $(t_1, r) \in \tilde{\mathsf{wr}}_x$; t_1 is committed. Hence, by the definitions of $\mathsf{readFrom}$ and $\mathsf{snapshot}_\iota$ on Figure A.3 and the rules semantics on Figure A.2, we deduce that $\mathbf{T}_\rho(\mathtt{commit}(t_1)) > \mathbf{T}_\rho(\mathtt{begin}(t_2))$. Next, as $\mathtt{last}(h, j)$ is pending in h, it is $\mathsf{so} \cup \mathsf{wr}$-maximal. Therefore, it is also $\mathsf{so}' \cup \mathsf{wr}'$-maximal; so it cannot play the role of t_1. However, it can play the role of t_2, $\mathtt{last}(h, j)$ or the role of an intermediate transaction. Let us analyze case by case depending on the axiom:
  - $\underline{a = \mathsf{Serializability}}$: Two sub-cases arise:
    - ∗ $\underline{\mathtt{last}(h, j) = t_2}$: I this case, t_2 writes x must hold. As ρ' is a prefix of a total run ρ^T, there exists runs $\hat{\rho}, \hat{\rho}'$ s.t. $\mathsf{rule}(\hat{\rho}, j', \hat{\rho}')$ is either COMMIT

    or ABORT and both a prefix of ρ^T; where j' is the session of $\mathsf{tr}(r)$. Without loss of generality, we can assume that $\hat{\rho}$ and $\hat{\rho}'$ have minimal size; so $\mathtt{last}(\mathsf{history}(\hat{\rho}), j') = \mathsf{tr}(r)$. As ρ^T is total and $\hat{\rho}'$ is a prefix of ρ^T, $\mathsf{validate}_\iota(\mathsf{history}(\hat{\rho}, \mathbf{T}_{\hat{\rho}'}, \mathsf{tr}(r)))$ holds. Note that as $(t_1, t_2) \in \mathsf{co}_{\rho'}$ and they are both committed, $\mathbf{T}_{\hat{\rho}'}(\mathtt{commit}(t_1)) < \mathbf{T}_{\hat{\rho}'}(\mathtt{commit}(t_2))$. However, $\mathsf{tr}(r)$ reads x, t_2 writes x and $\mathbf{T}_{\hat{\rho}'}(\mathtt{begin}(\mathsf{tr}(r))) < \mathbf{T}_{\hat{\rho}'}(\mathtt{commit}(\mathtt{last}(h, j))) < \mathbf{T}_{\hat{\rho}'}(\mathtt{commit}(\mathsf{tr}(r)))$; which contradicts that $\mathsf{validate}_\iota(\mathsf{history}(\hat{\rho}, \mathbf{T}_{\hat{\rho}'}, \mathsf{tr}(r)))$ holds. In conclusion, this case is impossible.

    - $\underline{\mathtt{last}(h, j) = \mathsf{tr}(r)}$: In such case, as t_1 and t_2 are committed, $(t_2, \mathtt{last}(h, j)) \in \mathsf{co}_\rho$ and $(t_1, t_2) \in \mathsf{co}_\rho$. Hence, this case is also impossible as co_ρ witnesses that h is consistent.

- $\underline{a = \mathsf{Prefix}}$: In this case, there exists a transaction t_4 s.t. $(t_2, t_4) \in \mathsf{co}^*_{\rho'}$ and $(t_4, \mathsf{tr}(r)) \in \mathsf{so}' \cup \mathsf{wr}'$. As $\mathtt{last}(h, j)$ is pending in h, by Lemma 1, $(\mathsf{so} \cup \mathsf{wr})$-maximal. Thus, as $\mathsf{so}' = \mathsf{so}$ and $\mathsf{wr}' = \mathsf{wr}$, $t_4 \neq \mathtt{last}(h, j)$. Moreover, as $(t_2, t_4) \in {\mathsf{co}_{\rho'}}^*$, t_4 is committed and $\mathtt{last}(h, j) \neq t_4$ is the $\mathsf{co}_{\rho'}$-maximal transaction that is committed, $t_2 \neq \mathtt{last}(h, j)$. Hence, $\mathtt{last}(h, j) = \mathsf{tr}(r)$. However, as $\mathsf{so}' = \mathsf{so}$, $\mathsf{wr}' = \mathsf{wr}'$ and $\mathsf{co}_{\rho'} \restriction_{T \setminus \{\mathtt{last}(h,j)\} \times T \setminus \{\mathtt{last}(h,j)\}} = \mathsf{co}_\rho \restriction_{T \setminus \{\mathtt{last}(h,j)\} \times T \setminus \{\mathtt{last}(h,j)\}}$; we conclude that $(t_1, t_2) \in \mathsf{co}_\rho$, $(t_2, t_4) \in {\mathsf{co}_\rho}^*$ and $(t_4, \mathtt{last}(h, j)) \in \mathsf{so} \cup \mathsf{wr}$; which contradicts that co_ρ witnesses h's consistency, so this case is impossible.
- $\underline{a = \mathsf{Conflict}}$: In this case, there exists a variable y and a transaction t_4 s.t. t_4 writes y, $\mathsf{tr}(r)$ writes y $(t_2, t_4) \in \mathsf{co}^*_{\rho'}$, $(t_4, \mathsf{tr}(r)) \in \mathsf{co}_{\rho'}$. As $\mathtt{last}(h, j)$ is the $\mathsf{co}_{\rho'}$-maximal transaction that is committed, $(t_2, \mathsf{tr}(r)), (t_4, \mathsf{tr}(r)) \in \mathsf{co}_{\rho'}$ and $\mathsf{writes}(\mathsf{tr}(r)) \neq \emptyset$, we deduce that $\mathtt{last}(h, j) \neq t_2, t_4$. Hence, $\mathtt{last}(h, j)$ must be $\mathsf{tr}(r)$ and $e = \mathtt{commit}(\mathtt{last}(h, j))$. On one hand, we observe that as $(t_4, \mathtt{last}(h, j)) \in \mathsf{co}_{\rho'}$ and they are both committed, $\mathbf{T}_{\rho'}(\mathtt{commit}(t_4)) < \mathbf{T}_{\rho'}(e)$. On the other hand, as $(t_2, t_4) \in \mathsf{co}^*_{\rho'}$ and $\mathbf{T}_{\rho'}(\mathtt{begin}(\mathsf{tr}(r))) < \mathbf{T}_{\rho'}(\mathtt{commit}(t_2))$; we conclude that $\mathbf{T}_{\rho'}(\mathtt{begin}(\mathsf{tr}(r))) < \mathbf{T}_{\rho'}(\mathtt{commit}(t_4))$. In conclusion, we obtain that $\mathsf{validate}_{\mathtt{SI}}(h', \mathbf{T}_{\rho'}, \mathtt{last}(h, j))$ does not hold due to the existence of t_4; which contradict the hypothesis, so this case is impossible.
- $\underline{a = \mathsf{Read\ Committed}}$: In this case, $r \neq e$ as r is a read event and e is not, and $(t_2, r) \in (\mathsf{so}' \cup \mathsf{wr}'); \mathsf{po}'^*$. Hence, as $\mathsf{so}' = \mathsf{so}, \mathsf{wr}' = \mathsf{wr}$ and $\mathsf{po}' = \mathsf{po} \cup \{(e', e) \mid e' \in \mathtt{last}(h, j)\}$; $(t_2, r) \in (\mathsf{so} \cup \mathsf{wr}); \mathsf{po}^*$. Finally, as $\mathtt{last}(h, j)$ is pending in h, $\mathtt{last}(h, j) \neq t_2$. Thus, as $\mathsf{co}_{\rho'} \restriction_{T \setminus \{\mathtt{last}(h,j)\} \times T \setminus \{\mathtt{last}(h,j)\}} = \mathsf{co}_\rho \restriction_{T \setminus \{\mathtt{last}(h,j)\} \times T \setminus \{\mathtt{last}(h,j)\}}$; we deduce that $(t_1, t_2) \in \mathsf{co}_\rho$. However, this contradicts that co_ρ witnesses h's consistency; so this case is also impossible.

As every possible case is impossible, we deduce that the hypothesis, $\mathsf{co}_{\rho'}$ does not witnesses h''s consistency is false; so we conclude the proof of the inductive step.

□

B Proofs of Theorems 2 to 5

B.1 Complexity analysis of Algorithms 1 and 2 (Proof of Theorem 2)

For a given history h, Algorithm 1 computes necessary and sufficient conditions for an execution of h $\xi = (h, \mathsf{co})$ to be consistent. It computes a bigger relation $\mathsf{pco}_{\mathtt{res}}$ that includes co and any other dependency between transactions that can be deduced from the isolation configuration. Algorithm 1 decides if co is a commit order witnessing consistency of the history (Lemma 5) and it runs in polynomial time (Lemma 7).

Lemma 5. *For any full history $h = (T, \mathsf{so}, \mathsf{wr})$, the execution $\xi = (h, \mathsf{co})$ is consistent if and only if $\mathsf{pco}_{\mathtt{res}} = \textsc{saturate}(h, \mathsf{co})$ is acyclic.*

Proof. Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a history, $\xi = (h, \mathsf{co})$ be an execution of h and $\mathsf{pco}_{\mathtt{res}} = \textsc{saturate}(h, \mathsf{co})$ be the relation obtained thanks to Algorithm 1.

$\underline{\implies}$ Let us suppose that ξ is consistent. As co is acyclic, it suffice to prove that $\mathsf{pco}_{\mathtt{res}} = \mathsf{co}$. By contradiction, let us suppose that $\mathsf{pco}_{\mathtt{res}} \neq \mathsf{co}$. As $\mathsf{co} \subseteq \mathsf{pco}_{\mathtt{res}}$ (line 2), there exists t_1, t_2 s.t. $(t_2, t_1) \in \mathsf{pco}_{\mathtt{res}} \setminus \mathsf{co}$. In such case, such tuple must be added in line 8. Hence, there exists $x \in \mathsf{Keys}, e \in \mathsf{reads}(h)$ and $v \in \mathsf{vis}(\mathsf{iso}(h)(\mathsf{tr}(r)))$ s.t. $t_1 = \mathsf{wr}_x^{-1}(r)$ and $\mathsf{vis}_a^{\mathsf{co}}(t_2, r, x)$ holds in h. As ξ is consistent, $(t_2, t_1) \in \mathsf{co}$; which is impossible. Hence, $\mathsf{pco}_{\mathtt{res}} = \mathsf{co}$.

$\underline{\impliedby}$ Let us suppose that $\mathsf{pco}_{\mathtt{res}}$ is acyclic. By contradiction, let us suppose that ξ is not consistent. Then, there exists an read event r s.t. $C^{\mathsf{co}}_{\mathsf{iso}(h)(\mathsf{tr}(r))}(r)$ does not hold. Hence, by Equation (1), there exists $v \in \mathsf{vis}(\mathsf{iso}(h)(\mathsf{tr}(r)))$, $x \in \mathsf{Keys}$, $t_2 \in T$ s.t. $\mathsf{v}(\mathsf{co})(t_2, r, x)$ hold in h but $(t_2, t_1) \notin \mathsf{co}$; where $t_1 = \mathsf{wr}_x^{-1}(r)$. In such case, Algorithm 1 ensures in line 8 that $(t_2, t_1) \in \mathsf{pco}_{\mathtt{res}}$. However, as $\mathsf{co} \subseteq \mathsf{pco}_{\mathtt{res}}$ (line 2), co is a total order and $\mathsf{pco}_{\mathtt{res}}$ is acyclic, $\mathsf{co} = \mathsf{pco}_{\mathtt{res}}$. Thus, $(t_2, t_1) \in \mathsf{co}$; which is impossible. Thus, ξ is consistent. □

Lemma 6. *Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a history s.t. $\mathsf{iso}(h)$ is bounded by $k \in \mathbb{N}$, $x \in \mathsf{Keys}$ be a key, $t \in T$ be a transaction, r be a read event, $\mathsf{pco} \subseteq T \times T$ be a partial order and v be a visibility relation in $\mathsf{vis}(\mathsf{iso}(h)(\mathsf{tr}(r)))$. Evaluating $\mathsf{v}(\mathsf{pco})(t, r, x)$ is in $\mathcal{O}(|h|^{k-2})$.*

Proof. As $\mathsf{iso}(h)$ is bounded, there exists $k \in \mathbb{N}$ s.t. $|\mathsf{vis}(\mathsf{iso}(h)(t))| \leq k$. Hence, the number of quantifiers employed by a visibility relation is at most k (and at least 3 according to Equation 1). In addition, for each $\mathsf{v} \in \mathsf{vis}(\mathsf{iso}(h)(t))$ evaluating each condition $\mathsf{v}(\mathsf{pco})(t, r, x)$ can be modelled with an algorithm that employ $k-3$ nested loops, one per existential quantifier employed by v, and that for each quantifier assignment evaluates the quantifier-free part of the formula.

First, we observe that as WrCons predicate only query information about the $k-1$ quantified events, the size of such sub-formula is in $\mathcal{O}(k)$. Next, we notice that as $\mathtt{WHERE}$ predicate can be evaluated in constant time, for every key x and event w, computing $\mathtt{value}_{\mathsf{wr}}(x, w)$ is in $\mathcal{O}(k \cdot T)$. Hence, as k is constant, evaluating the quantifier-free formula of v is in $\mathcal{O}(|h|)$ and thus, evaluating $\mathsf{v}(\mathsf{pco})(t, r, x)$ is in $\mathcal{O}(|h|^{k-3} \cdot |h|) = \mathcal{O}(|h|^{k-2})$. □

Lemma 7. *Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a full history, k be a bound on $\mathsf{iso}(h)$ and $\mathsf{pco} \subseteq T \times T$ be a partial order. Algorithm 1 runs in $\mathcal{O}(|h|^{k+1})$.*

Proof. Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a full history. Algorithm 1 can be decomposed in two blocks: lines 4-8 and lines 6-8. Hence, the cost of Algorithm 1 is in $\mathcal{O}(|\mathsf{Keys}| \cdot |\mathsf{events}(h)| \cdot |T| \cdot U)$; where U is an upper-bound of the cost of evaluating lines 6-8. On one hand, both $|\mathsf{Keys}|$, $|\mathsf{events}(h)|$ and $|T|$ are in $\mathcal{O}(|h|)$. On the other hand, as $\mathsf{iso}(h)$ is bounded by k, by Lemma 6, $U \in \mathcal{O}(|h|^{k-2})$. Altogether, we deduce that Algorithm 1 runs in $\mathcal{O}(|h|^{k+1})$. □

Algorithm 2 generalizes the results for RA and RC in [7] for full histories with heterogeneous saturable isolation configurations; proving that such histories can be checked in polynomial time.

Theorem 2. *Checking consistency of full histories with bounded saturable isolation configurations can be done in polynomial time.*

We split the proof of Theorem 2 in two Lemmas: Lemma 8 that proves the correctness of Algorithm 2 and Lemma 9 that ensures its polynomial-time behavior.

Lemma 8. *For every full history $h = (T, \mathsf{so}, \mathsf{wr})$ whose isolation configuration is saturable, Algorithm 2 returns* true *if and only if h is consistent.*

Proof. Let $h = (T, \mathsf{so}, \mathsf{wr})$ a full history whose isolation configuration is saturable and let pco be the visibility relation defined in line 3 in Algorithm 2.

On one hand, let suppose that h is consistent and let $\xi = (h, \mathsf{co})$ be a consistent execution of h. If we show that $\mathsf{pco} \subseteq \mathsf{co}$, we can conclude that Algorithm 2 returns true as co is acyclic. Let $(t_2, t_1) \in \mathsf{pco}$ and let us prove that $(t_2, t_1) \in \mathsf{co}$. As $\mathsf{so} \cup \mathsf{wr} \subseteq \mathsf{co}$, by the definition of commit order, we can assume that $(t_2, t_1) \in \mathsf{pco} \setminus (\mathsf{so} \cup \mathsf{wr})$. In such case, there must exists $x \in \mathsf{Keys}, e \in \mathsf{reads}(h)$ and $\mathsf{v} \in \mathsf{vis}(\mathsf{iso}(h)(\mathsf{tr}(e)))$ s.t. t_2 writes x and $\mathsf{v}((\mathsf{so} \cup \mathsf{wr})^+)(t_2, e, x)$ holds. As $\mathsf{iso}(h)(\mathsf{tr}(e))$ is saturable, $\mathsf{v}((\mathsf{so} \cup \mathsf{wr})^+)(t_2, e, x)$ holds. Hence, as co is a commit order and $(\mathsf{so} \cup \mathsf{wr})^+ \subseteq \mathsf{co}$; $\mathsf{v}(\mathsf{co})(t_2, e, x)$ also holds. Therefore, as co witnesses h's consistency, we deduce that $(t_2, t_1) \in \mathsf{co}$.

On the other hand, let us suppose that Algorithm 2 returns true. Then, pco must be acyclic by the condition in line 4. Therefore, as pco is acyclic it can be extended to a total order co. Let us prove that the execution $\xi = (h, \mathsf{co})$ is consistent. Let $x \in \mathsf{Keys}, t_2 \in T, e \in \mathsf{reads}(h)$ and $\mathsf{v} \in \mathsf{vis}(\mathsf{iso}(h)(\mathsf{tr}(e)))$ s.t. t_2 writes x and $\mathsf{v}(\mathsf{co})(t_2, e, x)$ holds. As Algorithm 2 returns true, we deduce that Algorithm 1 checks the condition at line 7. As $\mathsf{iso}(h)(\mathsf{tr}(e))$ is saturable, $\mathsf{v}((\mathsf{so} \cup \mathsf{wr})^+)(t_2, e, x)$ also holds. Thus, $(t_2, t_1) \in \mathsf{pco}$. As $\mathsf{pco} \subseteq \mathsf{co}$, $(t_2, t_1) \in \mathsf{co}$; so co witnesses h's consistency.

□

Lemma 9. *For every full history h whose isolation configuration is bounded, Algorithm 2 runs in polynomial time with respect $\mathcal{O}(|h|)$.*

Proof. Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a full history whose isolation configuration is saturable. First, we observe that checking if a graph $G = (V, E)$ is acyclic can be easily done with a DFS in $\mathcal{O}(|V| + |E|)$. Thus, the cost of checking acyclicity of both $\mathsf{so} \cup \mathsf{wr}$ (line 2) and pco (line 4) is in $\mathcal{O}(|T| + |T|^2) = \mathcal{O}(|T|^2) \subseteq \mathcal{O}(|h|^2)$. Furthermore, by Lemma 7, the cost of executing Algorithm 1 is in $\mathcal{O}(|h|^{k+1})$; where k is a bound in $\mathsf{iso}(h)$. Thus, checking h's consistency with Algorithm 2 can be done in polynomial time. □

B.2 Proof of Theorem 3

Theorem 3. *Checking consistency of bounded-width client histories with bounded isolation configuration stronger than* $\mathtt{RC}$ *and* $\mathtt{width}(h) \geq 3$ *is NP-complete.*

The proof of Theorem 3 is structured in two parts: proving that the problem is in NP and proving that is NP-hard. The first part corresponds to Lemma 10; which is analogous as the proof of Lemma 18. The second part, based on a reduction to 1-in-3 SAT problem, corresponds to Lemmas 11, 13 and 17.

Lemma 10. *The problem of checking consistency for a bounded width client history* h *with an isolation configuration stronger than* $\mathtt{RC}$ *and* $\mathtt{width}(h) \geq 3$ *is in NP.*

Proof. Let $h = (T, \mathsf{so}, \mathsf{wr})$ a client history whose isolation configuration is stronger than $\mathtt{RC}$. Guessing a witness of h, $\overline{h}$, and an execution of $\overline{h}$, $\xi = (\overline{h}, \mathsf{co})$, can be done in $\mathcal{O}(|\mathsf{Keys}| \cdot |\mathsf{events}(h)|^2 + |T|^2) \subseteq \mathcal{O}(|h|^3)$. By Lemma 5, checking if ξ is consistent is equivalent as checking if $\textsc{saturate}(h', \mathsf{co})$ is an acyclic relation. As by Lemma 7, Algorithm 1 requires polynomial time, we conclude the result. □

For showing NP-hardness, we will reduce 1-in-3 SAT to checking consistency. Let φ be a boolean formula with n clauses and m variables of the form $\varphi = \bigwedge_{i=1}^{n} (v_i^0 \vee v_i^1 \vee v_i^2)$; we construct a history h_φ s.t. h_φ is consistent if and only if φ is satisfiable with exactly only one variable assigned the value $\mathtt{true}$. The key idea is designing a history with width 3 that is stratified in *rounds*, one per clause. In each round, three transactions, one per variable in the clause, "compete" to be first in the commit order. The one that precedes the other two correspond to the variable in φ that is satisfied.

First, we define the round 0 corresponding to the variables of φ. For every variable $x_i \in \mathtt{var}(\varphi), 1 \leq i \leq m$ we define an homonymous key x_i that represents such variable. Doing an abuse of notation, we say that $x_i \in \mathtt{var}(\varphi)$. Then, we create two transactions 1_i and 0_i associated to the two states of x_i, 1 and 0. The former contains the event $\mathtt{INSERT}(\{x_i : 1, 1_i : 1\})$ while the latter $\mathtt{INSERT}(\{x_i : 0, 0_i : 1\})$. Both 1_i and 0_i write also on a special key named 1_i and 0_i respectively to indicate on the database that they have committed.

Next, we define rounds $1 - n$ representing each clause in φ. For each clause $C_i \coloneqq (v_i^0 \vee v_i^1 \vee v_i^2), 1 \leq i \leq n$, we define the round i. Round i is composed by three transactions: t_i^0, t_i^1 and t_i^2, representing the choice of the variable among v_i^0, v_i^1 and v_i^2 that is selected in the clause C_i. Transactions t_i^j write on keys v_i^j and $v_i^{j+1 \bmod 3}$ to preserve the structure of the clause C_i, as well on the special homonymous key t_i^j to indicate that such transaction has been executed; in a similar way as we did in the round 0. For that, we impose that transactions t_i^j are composed of an event $\mathtt{SELECT}(\lambda x : \mathsf{eq}(x, v_i^j, v_i^{j+1 \bmod 3}, v_i^{j+2 \bmod 3}))$ followed by an event $\mathtt{INSERT}(\{v_i^j : 0, v_i^{j+1 \bmod 3} : 1, t_i^j : -1\})$.

The function $\mathtt{eq} : \mathsf{Rows} \times \mathsf{Keys}^3 \to \{\mathsf{true}, \mathsf{false}\}$ is described in Equation 6 and assumes that Rows contains two distinct values 0 and 1 and that there is a predicate $\mathtt{val} : \mathsf{Rows} \to \{0, 1\}$ that returns the value of a variable in the database. Intuitively, for any key r, if a, b, c correspond to the three variables in a clause C_i (possibly permuted), whenever $\neg\mathtt{eq}(r, a, b, c)$ holds, we deduce that the value assigned at key a is 1 while on the other two keys the assigned value is 0. Moreover, whenever r refers to any of the special keys such as $0_i, 1_i$ or t_i^j, the predicate $\mathtt{eq}(r, a, b, c)$ always holds.

$$\mathtt{eq}(r, a, b, c) = \begin{cases} \mathtt{val}(r) \neq 1 \text{ if } \mathsf{key}(r) = a \\ \mathtt{val}(r) \neq 0 \text{ if } \mathsf{key}(r) = b \vee \mathsf{key}(r) = c \\ \mathsf{true} & \text{if } \mathsf{key}(r) \in \{t_i^j \mid 1 \leq i \leq n, 0 \leq j \leq 2\} \\ \mathsf{true} & \text{if } \mathsf{key}(r) \in \{1_i, 0_i \mid 1 \leq i \leq m\} \\ \mathsf{false} & \text{otherwise} \end{cases} \tag{6}$$

Finally, we add an initial transaction that writes on every key the value 1. For that, we assume that Keys contains only one key per variable used in φ as well as one key per aforementioned transaction. We denote by T the set of all described transactions as well as by $\mathtt{round}(t)$ to the round a transaction $t \in T$ belongs to.

We describe the session order in the history h_φ using an auxiliary relation $\overline{\mathsf{so}}$. We establish that $(1_i, 1_j), (0_i, 0_j) \in \overline{\mathsf{so}}$ for any pair of indices $i, j, 1 \leq i < j \leq m$. We also enforce that $(t_i^j, t_{i+1}^j) \in \overline{\mathsf{so}}$, for every $1 \leq i \leq n, 0 \leq j, j' \leq 2$. Finally, we connect round 0 with round 1 by enforcing that $(1_m, t_1^0) \in \overline{\mathsf{so}}$ and $(0_m, t_1^1) \in \overline{\mathsf{so}}$. Then, we denote by so to the transitive closure of $\overline{\mathsf{so}}$. Note that so is a union of disjoint total orders, so it is acyclic.

For describing the write-read relation, we distinguish between two cases: keys associated to variables in φ or to a transaction in T. On one hand, for every key $x_i, 1 \leq i \leq m$, we define $\mathsf{wr}_{x_i} = \emptyset$. On the other hand, for every key x associated to a transaction t_x and every read event r in a transaction t, we impose that $(t_x, r) \in \mathsf{wr}_x$ if $\mathtt{round}(t_x) < \mathtt{round}(t)$ while otherwise we declare that $(\mathtt{init}, r) \in \mathsf{wr}_x$. Then, we denote by $\mathsf{wr} = \bigcup_{x \in \mathsf{Keys}} \mathsf{wr}_x$ as well as by h_φ to the tuple $h_\varphi = (T, \mathsf{so}, \mathsf{wr})$. A full depiction of h_φ can be found in Figure B.1.

We observe that imposing $\mathsf{wr}_x = \emptyset$ on every key $x \in \mathtt{var}(\varphi)$ ensures that, for any witness of h_φ, $\overline{h} = (T, \mathsf{so}, \overline{\mathsf{wr}})$, if $(w, r) \in \overline{\mathsf{wr}}$, then $\mathtt{WHERE}(r)(\mathtt{value}_{\overline{\mathsf{wr}}}(w, x)) = 0$. In particular, this implies that each transaction t_i^j must read key v_i^j from a transaction that writes 1 as value while it also must read keys $v_i^{j+1 \bmod 3}$ and $v_i^{j+2 \bmod 3}$ from a transaction that writes 0 as value. Intuitively, this property shows that φ is well-encoded in h_φ.

The proof is divided in four steps: Lemma 11 proves that the h_φ is a polynomial-size transformation of φ, Lemma 12 proves that the h_φ is indeed a history and Lemmas 13 and 17 prove that h_φ is consistent if and only if φ is 1-in-3 satisfiable.

Lemma 11. *h_φ is a polynomial size transformation on the length of φ.*

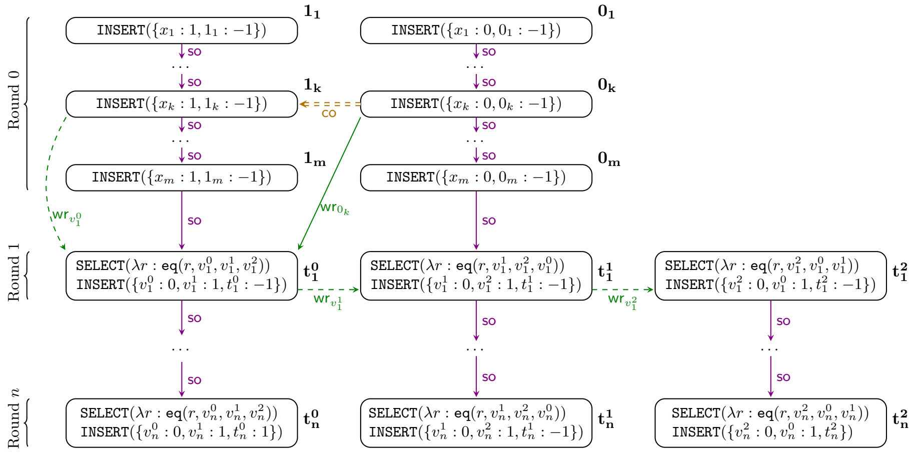

Round 0
Round 1
Round n
$\mathbf{1_1}$
INSERT($\{x_1 : 1, 1_1 : -1\}$)
$\mathbf{0_1}$
INSERT($\{x_1 : 0, 0_1 : -1\}$)
so
$\mathbf{1_k}$
INSERT($\{x_k : 1, 1_k : -1\}$)
co
$\mathbf{0_k}$
INSERT($\{x_k : 0, 0_k : -1\}$)
$\mathbf{1_m}$
INSERT($\{x_m : 1, 1_m : -1\}$)
$\mathbf{0_m}$
INSERT($\{x_m : 0, 0_m : -1\}$)
$\mathsf{wr}_{v_1^0}$
wr_{0_k}
SELECT(λr : eq(r, v_1^0, v_1^1, v_1^2))
INSERT($\{v_1^0 : 0, v_1^1 : 1, t_1^0 : -1\}$)
$\mathbf{t_1^0}$
$\mathsf{wr}_{v_1^1}$
SELECT(λr : eq(r, v_1^1, v_1^2, v_1^0))
INSERT($\{v_1^1 : 0, v_1^2 : 1, t_1^1 : -1\}$)
$\mathbf{t_1^1}$
$\mathsf{wr}_{v_1^2}$
SELECT(λr : eq(r, v_1^2, v_1^0, v_1^1))
INSERT($\{v_1^2 : 0, v_1^0 : 1, t_1^2 : -1\}$)
$\mathbf{t_1^2}$
SELECT(λr : eq(r, v_n^0, v_n^1, v_n^2))
INSERT($\{v_n^0 : 0, v_n^1 : 1, t_n^0 : 1\}$)
$\mathbf{t_n^0}$
SELECT(λr : eq(r, v_n^1, v_n^2, v_n^0))
INSERT($\{v_n^1 : 0, v_n^2 : 1, t_n^1 : -1\}$)
$\mathbf{t_n^1}$
SELECT(λr : eq(r, v_n^2, v_n^0, v_n^1))
INSERT($\{v_n^2 : 0, v_n^0 : 1, t_n^2\}$)
$\mathbf{t_n^2}$

Fig. B.1: Description of the history h_φ from Theorem 3. Dashed edges only belong to a possible consistent witness of h_φ, where we assume $v_1^0 = x_k$. Transaction t_1^0 reads v_1^0, v_1^1 and v_1^2 from round 0; imposing some constraints on the transactions that write them. Due to axiom RC's definition, transaction t_1^1 must read v_1^1 from t_1^0 while transaction t_1^2 must read v_1^1 from t_1^1.

Proof. If φ has n clauses and m variables, h_φ employs $6n+2m+1$ transactions. As $m \leq 3n$, $|T| \in \mathcal{O}(n)$. The number of variables, $|\mathsf{Keys}| = m + |T|$, so $|\mathsf{Keys}| \in \mathcal{O}(n)$. As every transaction has at most two events, $|\mathsf{events}(h_\varphi)| \in \mathcal{O}(n)$. Moreover, $\mathsf{wr} \subseteq \mathsf{Keys} \times T \times T$ and $\mathsf{so} \subseteq T \times T$, so $|\mathsf{wr}| \in \mathcal{O}(n^3)$ and $|\mathsf{so}| \in \mathcal{O}(n^2)$. Thus, h_φ is a polynomial transformation of φ. □

For proving that h_φ is a history, by Definition 1 it suffices to prove that $\mathsf{so} \cup \mathsf{wr}$ is an acyclic relation. Indeed, by our choice of wr, for every key x, wr_x^{-1} is a partial function that, whenever it is defined, associates reads to writes on x. Hence, from Lemma 12 we conclude that h_φ is a history.

Lemma 12. *The relation* $\mathsf{so} \cup \mathsf{wr}$ *is acyclic.*

Proof. For proving that $\mathsf{so} \cup \mathsf{wr}$ is acyclic, we reason by induction on the number of clauses. In particular, we show that for every pair of transactions t, t' if $\texttt{round}(t') \leq i$ and $(t, t') \in \mathsf{so} \cup \mathsf{wr}$, then $\texttt{round}(t) \leq i$ and $(t', t) \notin \mathsf{so} \cup \mathsf{wr}$.

- Base case: The base case refers to round 0; which contains `init` and transactions $0_j, 1_j, 1 \leq j \leq m$. We observe that transactions in round 0 do not contain any read event. Hence, $(t, t') \in \mathsf{so}$. In such case, the result immediately holds by construction of so.
- Inductive case: Let us suppose that the induction hypothesis holds for every $1 \leq i \leq k \leq n$ and let us prove it also for $k + 1 \leq n$. If $\texttt{round}(t') < k + 1$, $\texttt{round}(t') \leq k$ and the result holds by induction hypothesis; so we can assume without loss of generality that $\texttt{round}(t') = k + 1$. By construction of both

so and wr, if $(t, t') \in \mathsf{so} \cup \mathsf{wr}$, $\mathtt{round}(t) < \mathtt{round}(t')$. Hence, $\mathtt{round}(t) \leq k$. By induction hypothesis on t, if $(t', t) \in \mathsf{so} \cup \mathsf{wr}$, $\mathtt{round}(t') \leq k < k + 1 = \mathtt{round}(t')$; which is impossible. Thus, we conclude that $(t', t) \notin \mathsf{so} \cup \mathsf{wr}$.

□

Lemma 13. *If φ is* 1*-in-*3 *satisfiable then h_φ is consistent.*

Proof. Let $\alpha : \mathsf{Keys} \to \{0, 1\}$ an assignment that makes 1-in-3 satisfiable. To construct a witness of h_φ we define a write-read relation $\overline{\mathsf{wr}}$ that extends wr and a total order on its transactions. For that, we first define a total order co between the transactions in T. In Equation 6 we define two auxiliary relations $\hat{\mathtt{r}}$ and $\hat{\mathtt{b}}$ based on α that totally orders the transactions that belongs to the same round.

For every clause $C_i, 1 \leq i \leq n$ let j_i be the unique index s.t. $\alpha(v_i^{j_i}) = 1$. Such index allow us to order the transactions in the round i: $t_i^{j_i}$ preceding $t_i^{j_i+1 \bmod 3}$ while $t_i^{j_i+1 \bmod 3}$ preceding $t_i^{j_i+2 \bmod 3}$. Intuitively, $t_i^{j_i}$ must precede the other two transactions in the total order as v_i^j is the variable that is satisfied. Then, we connect every pair of consecutive rounds thanks to relation $\hat{\mathtt{c}}_1$.

For transactions in round 0, we enforce that transactions associated to the same variable are totally ordered using α. In particular, for every $i, 1 \leq i \leq m$, 0_i precedes 1_i in $\hat{\mathtt{b}}$ if and only if $\alpha(v_i) = 1$. Then, we connect every pair tuple in $\hat{\mathtt{b}}$ with relation $\hat{\mathtt{c}}_2$. Finally, we connect $\mathtt{init}$ with transactions in round 0 as well as round 0 with round 1 thanks to relation $\hat{\mathtt{c}}_3$.

$$\begin{aligned}
\hat{\mathtt{r}} &= \left\{ \begin{array}{l} (t_i^{j_i}, t_i^{j_i+1 \bmod 3}) \\ (t_i^{j_i+1 \bmod 3}, t_i^{j_i+2 \bmod 3}) \end{array} \middle| \begin{array}{l} 1 \leq i \leq n, 0 \leq j_i \leq 2 \\ \alpha(v_i^{j_i}) = 1 \end{array} \right\} \\
\hat{\mathtt{b}} &= \{(0_i, 1_i) \mid x_i \in \mathsf{Keys} \wedge \alpha(x_i) = 1\} \cup \{(1_i, 0_i) \mid x_i \in \mathsf{Keys} \wedge \alpha(x_i) = 0\} \\
\hat{\mathtt{c}}_1 &= \{(t_i^{j_i+2 \bmod 3}, t_{i+1}^{j_{i+1}}) \mid 1 \leq i < n, 0 \leq j_i, j_{i+1} \leq 2, \alpha(v_i^{j_i}) = 1 = \alpha(v_i^{j_{i+1}})\} \\
\hat{\mathtt{c}}_2 &= \{(1_i, 0_j), (1_i, 1_j), (0_i, 0_j), (0_i, 1_j) \mid 1 \leq i < j \leq m\} \\
\hat{\mathtt{c}}_3 &= \{(\mathtt{init}, 0_1), (\mathtt{init}, 1_1)\} \cup \{(1_m, t_1^{j_1}), (0_m, t_1^{j_1}) \mid 0 \leq j_1 \leq 2, \alpha(v_i^{j_1}) = 1\}
\end{aligned} \tag{7}$$

Let $\mathsf{co} = (\hat{\mathtt{r}} \cup \hat{\mathtt{c}}_1 \cup \hat{\mathtt{b}} \cup \hat{\mathtt{c}}_2 \cup \hat{\mathtt{c}}_3)^+$. The proof of Lemma 13 concludes thanks to Lemmas 14 and 15, Proposition 1 and Corollary 1. First, Lemma 14 proves that the relation co is a total order between transactions. Then, Lemma 15 shows that co allow us define $\overline{h}$, a witness of h_φ. And finally, with the aid of Proposition 1 and Corollary 1 we conclude that $\overline{h}$ is consistent; so it is a consistent witness of h_φ.

□

Lemma 14. *The relation co is a total order.*

Proof. For proving that co is a total order, we show by induction that if $(t, t') \in \mathsf{co}$ and $\mathtt{round}(t') \leq i$, then $\mathtt{round}(t) \leq i$ and $(t', t) \notin \mathsf{co}$.

- Base case: We observe that by construction of co, $t' \neq \mathtt{init}$. We prove the base case by a second induction that if there exists $i', 1 \leq i' \leq m$ s.t. $t' \in \{0_{i'}, 1_{i'}\}$ and $(t, t') \in \mathsf{co}$ then either $t = \mathtt{init}$ or there exists $i \leq i'$ s.t. $t \in \{0_i, 1_i\}$ and $(t', t) \notin \mathsf{co}$.
  - Base case: Let us suppose that $\alpha(x_0) = 1$ as the other case is symmetric. If $t = \mathtt{init}$, $(t', t) \notin \mathsf{co}$ as $\mathtt{init}$ is minimal in co. If not, then $t' = 1_1$ and $t = 0_1$. We conclude once more that $(t, t') \notin \mathsf{co}$ as 0_1 only have $\mathtt{init}$ as a co-predecessor; which is co-minimal.
  - Induction hypothesis: Let us suppose that the induction hypothesis holds for every $1 \leq i \leq k \leq m$ and let us prove it also for $k + 1 \leq m$. If $i' < k$, we conclude the result by induction hypothesis; so we can assume that $i' = k$. Moreover, as $\mathtt{init}$ is co-minimal, we can assume without loss of generality that $t \neq \mathtt{init}$. Thus, by construction of co, there must exists $i, 1 \leq i \leq m$ s.t. $t \in \{0_i, 1_i\}$. In particular, $i \leq i'$. Thus, if $i < i'$ and (t', t) would be in co, by induction hypothesis on t we would deduce that $i' \leq i < i'$; which is impossible. Hence, we can assume that $i' = i$. Let us assume that $\alpha(x_i) = 1$ as the other case is symmetric. Thus, we deduce that $t = 0_i$ and $t' = 1_i$. We observe that $(t', t) \notin \hat{\mathtt{r}} \cup \hat{\mathtt{c}}_1 \cup \hat{\mathtt{b}} \cup \hat{\mathtt{c}}_2 \cup \hat{\mathtt{c}}_3$. As T is finite, if $(t', t) \in \mathsf{co}$, there would exist a transaction $t'' \neq t'$ s.t $(t'', t) \in \hat{\mathtt{r}} \cup \hat{\mathtt{c}}_1 \cup \hat{\mathtt{b}} \cup \hat{\mathtt{c}}_2 \cup \hat{\mathtt{c}}_3$ and $(t', t'') \in \mathsf{co}$. But in such case, either $t'' = \mathtt{init}$ or there would exists an integer $i'', 1 \leq i'' < i \leq m$ s.t. $t'' \in \{0_{i''}, 1_{i''}\}$; which is impossible by induction hypothesis. In conclusion, $(t', t) \notin \mathsf{co}$.
- Inductive case: Let us suppose that the induction hypothesis holds for every $1 \leq i \leq k \leq n$ and let us prove it also for $k+1 \leq n$. Let thus t, t' transactions s.t. $\mathtt{round}(t)' \leq k+1$ and $(t, t') \in \mathsf{co}$. If $\mathtt{round}(t') < k+1$, $\mathtt{round}(t') \leq k$ and the result holds by induction hypothesis; so we can assume without loss of generality that $\mathtt{round}(t') = k + 1$. By construction of co, $\mathtt{round}(t) \leq k + 1$. If $\mathtt{round}(t) \leq k$ and $(t', t) \in \mathsf{co}$, by induction hypothesis on t we obtain that $\mathtt{round}(t') \leq k < k + 1 = \mathtt{round}(t')$; which is impossible. Thus, we can also assume without loss of generality that $\mathtt{round}(t) = k + 1$. In such case, we observe that $\hat{\mathtt{c}}_1$, $\hat{\mathtt{b}}$ and $\hat{\mathtt{c}}_1$ do not order transactions belonging to the same round. Hence if $(t, t') \in \mathsf{co}$ and $(t', t) \in \mathsf{co}$, we deduce that $(t, t') \in \hat{\mathtt{r}}$ and $(t', t) \in \hat{\mathtt{r}}$. However, by construction of $\hat{\mathtt{r}}$, this is impossible, so we conclude once more that $(t', t) \notin \mathsf{co}$.

□

Next, we construct a full history $\overline{h}$ using co that extends h_φ. For every key x and $\mathtt{read}$ event r, we define w_x^r as follows:

$$w_x^r = \max_{\mathsf{co}}\{t \in T \mid t \ \mathsf{writes} \ x \wedge (t, r) \in \mathsf{co}\} \tag{8}$$

Observe that w_x^r is well-defined as co is a total order and $\mathtt{init}$ write every key. For each key $x \in \mathtt{var}(\varphi)$, we define the relation $\overline{\mathsf{wr}}_x = \{(w_x^r, r) \mid r \in \mathsf{reads}(h)\}$. Then, we define the relation $\overline{\mathsf{wr}} = \bigcup_{x \in \mathtt{var}(\varphi)} \overline{\mathsf{wr}}_x \cup \mathsf{wr}$ as well as the history

$\overline{h} = (T, \mathsf{so}, \overline{\mathsf{wr}})$. Lemma 15 proves that $\overline{h}$ is indeed a full history while Lemma 16 shows that $\overline{h}$ is a witness of h_φ.

Lemma 15. *$\overline{h}$ is a full history.*

Proof. For showing that $\overline{h}$ is a full history it suffices to show that $\mathsf{so} \cup \overline{\mathsf{wr}}$ is acyclic. As co is a total order and $\overline{\mathsf{wr}} \setminus \mathsf{wr} \subseteq \mathsf{co}$, proving that $\mathsf{so} \cup \mathsf{wr} \subseteq \mathsf{co}$ concludes the result. First, we prove that $\mathsf{so} \subseteq \mathsf{co}$. Let t, t' be transactions s.t. $(t, t') \in \mathsf{so}$. In such case, $\texttt{round}(t) \leq \texttt{round}(t')$; and they only coincide if $\texttt{round}(t) = \texttt{round}(t') = 0$. Three cases arise:

- $\underline{\texttt{round}(t) = \texttt{round}(t') = 0}$: As $(t, t') \in \hat{\mathtt{c}}_2$, we conclude that $(t, t') \in \mathsf{co}$.
- $\underline{\texttt{round}(t), \texttt{round}(t') > 0}$: As $\texttt{round}(t), \texttt{round}(t') > 0$ and $\texttt{round}(t) \leq \texttt{round}(t')$, by construction of so we deduce that $\texttt{round}(t) < \texttt{round}(t')$. As co is transitive, we can assume without loss of generality that $\texttt{round}(t') = \texttt{round}(t) + 1$. Therefore, there exists $i, j, 1 \leq i < n, 0 \leq j \leq 2$ s.t. $t = t_i^j$ and $t' = t_{i+1}^j$. Let $j_i, j_{i+1}, 0 \leq j_i, j_{i+1} \leq 2$ be the integers s.t. $\alpha(v_i^{j_i}) = 1 = \alpha(v_{i+1}^{j_{i+1}})$. In such case, we know that $(t_i^j, t_i^{j_i+2 \bmod 3}) \in \hat{\mathtt{r}}^*$, $(t_i^{j_i+2 \bmod 3}, t_{i+1}^{j_i}) \in \hat{\mathtt{c}}_1$ and $(t_{i+1}^{j_{i+1}}, t_{i+1}^{j+1}) \in \hat{\mathtt{r}}^*$. Hence, as co is transitive, $(t, t') \in \mathsf{co}$.
- $\underline{\texttt{round}(t) = 0, \texttt{round}(t') > 0}$: In this case, as $\texttt{round}(t) = 0$, there exists $i, 1 \leq i \leq m$ s.t. $x_i \in \texttt{var}(\varphi), t \in \{0_i, 1_i\}$. We assume without loss of generality that $t = 1_i$ as the other case is symmetric. In addition, as $\texttt{round}(t') > 0$ and $(t, t') \in \mathsf{so}$, there exists $i, 1 \leq i \leq n$ s.t. $t' = t_i^0$. We rely on the two previous proven cases to deduce the result: as $(0_i, 0_m) \in \mathsf{so} \subseteq \mathsf{co}$, $(0_m, t_0^{j_0}) \in \hat{\mathtt{c}}_3$, $(t_0^{j_0}, t_0^0) \in \hat{\mathtt{r}}^*$ and $(t_0^0, t_i^0) \in \mathsf{so} \subseteq \mathsf{co}$, we conclude that $(t, t') \in \mathsf{co}$.

Next, we prove that $\mathsf{wr} \subseteq \mathsf{co}$. Let r be a read event and w be a write event s.t. $(w, r) \in \mathsf{wr}$. Then, there exists $i, i', 1 \leq i < i' \leq n$ and $j, j', 0 \leq j, j' \leq 2$ s.t. $w = t_i^j$ and $\mathsf{tr}(r) = t_{i'}^{j'}$. Let $j_{i'-1}, j_{i'}, 0 \leq j_{i'-1}, j_{i'} \leq 2$ be the integers s.t. $\alpha(v_{i'-1}^{j_{i'-1}}) = 1 = \alpha(v_{i'}^{j'})$. In such case, we know that $(t_i^j, t_{i'-1}^j) \in \mathsf{so}^*$, $(t_{i'-1}^j, t_{i'-1}^{j_{i'}+2 \bmod 3}) \in \hat{\mathtt{r}}^*$, $(t_{i'-1}^{j_{i'}+2 \bmod 3}, t_{i'}^{j_i}) \in \hat{\mathtt{c}}_1$ and $(t_{i'}^{j_{i'}}, t_{i'}^{j'}) \in \hat{\mathtt{r}}^*$. As $\mathsf{so} \subseteq \mathsf{co}$ and co is transitive, we conclude that $(w, r) \in \mathsf{co}$. □

We show that $\overline{h}$ is indeed a full history, that is a witness of h_φ and that also witness h_φ's consistency.

Lemma 16. *The history $\overline{h}$ is a witness of h_φ.*

Proof. By Lemma 15 $\overline{h}$ is a full history. Hence, for proving that $\overline{h}$ is a witness of h_φ, we need to show that for every key $x \in \mathsf{Keys}$ and every read r, if $\mathsf{wr}_x^{-1}(r) \uparrow$, $\texttt{WHERE}(r)(\texttt{value}_{\overline{\mathsf{wr}}}(w_x^r, x)) = 0$. Note that by construction of h_φ, such cases coincide with $x \in \texttt{var}(\varphi)$. In addition, we observe that if r is a read event, there exists indices $1 \leq i \leq n, 0 \leq j \leq 2$ s.t. $r \in t_i^j$. Thus, by Equation 6, we only need to prove that $\texttt{WHERE}(r)(\texttt{value}_{\overline{\mathsf{wr}}}(w_x^r, x)) = 0$ whenever x is v_i^0, v_i^1 or v_i^2.

We prove as an intermediate step that in each round, every key has the same value in $\overline{h}$. For every round i and key $x \in \texttt{var}(\varphi)$, we consider the transaction

$t^i_x = \max_{\mathsf{co}}\{t \mid t \text{ } \mathsf{writes} \text{ } x \land \mathtt{round}(t) \leq i\}$. We prove by induction on the number of the round that $\mathtt{value}_{\overline{\mathsf{wr}}}(t^i_x, x) = \mathtt{value}_{\overline{\mathsf{wr}}}(t^0_x, x) = (x, \alpha(x))$.

- Base case: The base case, $i = 0$, is immediately trivial. Note that in this case $\mathtt{value}_{\overline{\mathsf{wr}}}(t^0_x, x) = \alpha(x)$.
- Inductive case: Let us assume that $\mathtt{value}_{\overline{\mathsf{wr}}}(t^{i-1}_x, x) = \mathtt{value}_{\overline{\mathsf{wr}}}(t^0_x, x)$ and let us prove that $\mathtt{value}_{\overline{\mathsf{wr}}}(t^i_x, x) = (x, \alpha(x))$. Note that in round i only keys v^0_i, v^1_i and v^2_i are written; so for every other key x, $t^i_x = t^{i-1}_x$ and by induction hypothesis, $\mathtt{value}_{\overline{\mathsf{wr}}}(t^i_x, x) = (x, \alpha(x))$. Let thus $j, 0 \leq j \leq 2$ s.t. $\alpha(v^j_i) = 1$. In this case, $t^i_{v^j_i} = t^i_{v^{j+2 \bmod 3}_i} = t^{j+2 \bmod 3}_i$ and $t^i_{v^{j+1 \bmod 3}_i} = t^{j+1 \bmod 3}_i$. Hence, we can conclude the inductive step as:

$$\mathtt{value}_{\overline{\mathsf{wr}}}(t^{j+2 \bmod 3}_i, v^j_i) = (v^j_i, 1) = (v^j_i, \alpha(v^j_i))$$
$$\mathtt{value}_{\overline{\mathsf{wr}}}(t^{j+1 \bmod 3}_i, v^{j+1 \bmod 3}_i) = (v^{j+1 \bmod 3}_i, 0) = (v^{j+1 \bmod 3}_i, \alpha(v^{j+1 \bmod 3}_i))$$
$$\mathtt{value}_{\overline{\mathsf{wr}}}(t^{j+2 \bmod 3}_i, v^{j+2 \bmod 3}_i) = (v^{j+2 \bmod 3}_i, 0) = (v^{j+2 \bmod 3}_i, \alpha(v^{j+2 \bmod 3}_i))$$

We can thus conclude that $\overline{h}$ is a witness of h_φ. Let $i, j, 1 \leq i \leq n, 0 \leq j \leq 2$ be indices s.t. $\alpha(v^j_i) = 1$. For simplifying notation, we denote by t_0, t_1, t_2 to the transactions $t^j_i, t^{j+1 \bmod 3}_i$ and $t^{j+2 \bmod 3}_i$ respectively. We also denote by r_0, r_1, r_2 to the read events that belong to t_0, t_1 and t_2 respectively as well by v_0, v_1, v_2 to the keys associated to t_0, t_1 and t_2 respectively. For every key $x \neq v_0, v_1, v_2$ and for every transaction t that writes x, $\mathtt{WHERE}(r_j)(\mathtt{value}_{\mathsf{wr}}(t, x)) = 0, 0 \leq j \leq 2$; so we can focus only on keys v_0, v_1 and v_2. Three cases arise:

- Transaction t_0: Let thus x be a key in $\{v_0, v_1, v_2\}$. By construction of h_φ and co, t_0 reads x from t^{i-1}_x. As proved before, $\mathtt{value}_{\overline{\mathsf{wr}}}(t^{i-1}_x, x) = (x, \alpha(x))$ and $\alpha(x) = (x, 1)$ if and only if $x = v_0$. Hence, as $\mathtt{WHERE}(r_0)(\mathtt{value}_{\overline{\mathsf{wr}}}(t^{i-1}_x, x)) = 0$ we conclude that $\mathtt{WHERE}(r_0)(\mathtt{value}_{\overline{\mathsf{wr}}}(w^{r_0}_x, x)) = 0$.
- Transaction t_1: In this case, t_1 reads v_2 from $t^{i-1}_{v_2}$ and it reads v_0 and v_1 from t_0. On one hand, $\mathtt{value}_{\overline{\mathsf{wr}}}(t^{i-1}_{v_2}, v_2) = (v_2, \alpha(v_2)) = (v_2, 0)$. Thus, as $\mathtt{WHERE}(r_1)(t^{i-1}_{v_2}) = 0$, we conclude that $\mathtt{WHERE}(r_1)(\mathtt{value}_{\overline{\mathsf{wr}}}(w^{r_1}_{v_2}, v_2)) = 0$. On the other hand, by construction of h_φ, $\mathtt{WHERE}(r_1)(\mathtt{value}_{\overline{\mathsf{wr}}}(t_0, v_0)) = \mathtt{WHERE}(r_1)(\mathtt{value}_{\overline{\mathsf{wr}}}(t_0, v_1)) = 0$. Thus, the result hold.
- Transaction t_2: In this case, t_2 read v_0 from t_0 and v_1 and v_2 from t_1. By construction of h_φ both $\mathtt{WHERE}(r_2)(\mathtt{value}_{\overline{\mathsf{wr}}}(t_0, v_0))$, $\mathtt{WHERE}(r_2)(\mathtt{value}_{\overline{\mathsf{wr}}}(t_1, v_1))$ and $\mathtt{WHERE}(r_2)(\mathtt{value}_{\overline{\mathsf{wr}}}(t_1, v_2))$ are equal to 0; so we conclude the result.

□

We conclude the proof showing that the execution $\xi = (\overline{h}, \mathsf{co})$ is a consistent execution of h_φ. We observe that by construction of $\overline{\mathsf{wr}}$ and co, $\overline{h}$ satisfies $\mathtt{SER}$ using co. Corollary 1 proves that $\mathsf{iso}(h_\varphi)$ is weaker than $\mathtt{SER}$; which allow us to conclude that so $\overline{h}$ satisfies $\mathsf{iso}(h_\varphi)$. In other words, that $\overline{h}$ is consistent.

Proposition 1. *Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a full history, $\xi = (h, \mathsf{co})$ be an execution of h, r be a* $\mathtt{read}$ *event, t_2 be a transaction distinct from $\mathsf{tr}(r)$, x be a key and $\mathsf{v} \in \mathsf{vis}(\mathsf{iso}(h)(\mathsf{tr}(e)))$. If $\mathsf{v}(t_2, r, x)$ holds in ξ, then $(t_2, \mathsf{tr}(r)) \in \mathsf{co}$.*

Proof. The proposition is result of an immediate induction on the definition of v. The base case is $\mathsf{po}, \mathsf{so}, \mathsf{wr} \subseteq \mathsf{co}$, which holds by definition. The inductive case follows from the operators employed in Equation 3: union, composition and transitive closure of relations; which are monotonic. □

As a consequence of Proposition 1 and Serializability axiom definition, we obtain the following result.

Corollary 1. *Any isolation level is weaker than* $\mathtt{SER}$*.*

Lemma 17. *If* h_φ *is consistent then* φ *is* 1*-in-*3 *satisfiable.*

Proof. If h_φ is consistent, there exists a consistent witness of h_φ $\overline{h} = (T, \mathsf{so}, \overline{\mathsf{wr}})$. As $\overline{h}$ is consistent and $\mathsf{iso}(h)$ is stronger than $\mathtt{RC}$, there exists a consistent execution of $\overline{h}$, $\xi = (\overline{h}, \mathsf{co})$. Let $\alpha_h : \mathtt{var}(\varphi) \to \{0, 1\}$ s.t. for every variable $v_j, 1 \leq j \leq m$, $\alpha_h(v_j) = 1$ if and only if $(0_j, 1_j) \in \mathsf{co}$. We show that φ is 1-in-3 satisfiable using α.

As an intermediate step, we prove that α_h describes the value read by any transaction in $\overline{h}$. For every $i, 0 \leq i \leq n$ and key $x \in \mathtt{var}(\varphi)$, let $t_x^i = \max_{\mathsf{co}}\{t \mid t \ \mathsf{writes}\ x \land \mathtt{round}(t) \leq i\}$. We prove by induction that for every $i, 0 \leq i \leq n$ (1) $\mathtt{value}_{\overline{\mathsf{wr}}}(t_x^i, x) = (x, \alpha(x))$, (2) for any $\mathtt{read}$ event r from a transaction t s.t. $\mathtt{round}(t) \leq i$, if $(w, r) \in \overline{\mathsf{wr}}_x$, then w coincides with $\max_{\mathsf{co}}\{t \in T \mid t \ \mathsf{writes}\ x \land (t, \mathsf{tr}(r)) \in \mathsf{co}\}$ and (3) if $i > 0$, $\alpha(v_i^j) = 1$ if and only if $(t_i^j, t_i^{j+1 \text{ mod } 3}) \in \mathsf{co}$ and $(t_i^{j+1 \text{ mod } 3}, t_i^{j+2 \text{ mod } 3}) \in \mathsf{co}$.

– Base case: Let $j, 1 \leq j \leq m$ be the integer s.t. $x = v_j$. In such case, (1) holds as $t_x^0 = 1_j$ if and only if $\alpha(v_j) = 1$; and in such case, $\mathtt{value}_{\overline{\mathsf{wr}}}(t_x^0, v_j) = (v_j, \alpha(v_j))$. Also (2) trivially holds as there is no $\mathtt{read}$ event in a transaction belonging to round 0. Finally, (3) also trivially holds as $i = 0$.
– Inductive case: We assume that (1), (2) and (3) hold for round $i - 1$ and let us prove it for round i. Let j the index of the co-minimal transaction among t_i^1, t_i^2, t_i^3. We denote by t_0, t_1, t_2 to $t_i^j, t_i^{j+1 \text{ mod } 3}$ and $t_i^{j+2 \text{ mod } 3}$ respectively, by r_0, r_1, r_2 to the unique $\mathtt{read}$ event in t_0, t_1 and t_2 respectively and by v_0, v_1 and v_2 to the keys associated to t_0, t_1 and t_2 respectively.
Let thus $x \in \mathtt{var}(\varphi)$ be a key, t be a transaction among t_0, t_1, t_2 and let w_x^t be a transaction s.t. $(w_x^t, t) \in \overline{\mathsf{wr}}_x$. As $\mathtt{round}(t_x^{i-1}) < \mathtt{round}(t)$, $(t_x^{i-1}, t) \in \mathsf{wr}_{t_x^{i-1}}$. Hence, as $\overline{h}$ satisfies $\mathtt{RC}$, either $w_x^t = t_x^{i-1}$ or $\mathtt{round}(w_x^t) = i$.
First we prove (3) analyzing t_0. As $(t_0, t_1) \in \mathsf{co}$ and $(t_0, t_2) \in \mathsf{co}$ and $\overline{\mathsf{wr}} \subseteq \mathsf{co}$ we deduce that $w_x^t = t_x^{i-1}$. In such case, as (1) holds by induction hypothesis and $\mathtt{WHERE}(r_0)(\mathtt{value}_{\overline{\mathsf{wr}}}(t_x^{i-1}, x)) = 0$, we conclude that $\alpha(x) = 1$ if $x = v_0$ and $\alpha(x) = 0$ if $x = v_1, v_2$.
For proving (2) we analyze three cases depending on t:
  - $t = t_0$: As proved before, if $t = t_0$, $w_x^t = t_x^{i-1}$. By definition of t_x^{i-1}, (2) holds.
  - $t = t_1$: As t_0 only writes v_0 and v_1 and $(t_1, t_2) \in \mathsf{co}$ we deduce that for every key $x \neq v_0, v_1$, $w_x^{t_1} = t_x^{i-1}$; which immediately implies (2). As (3) holds for round i, we know that $\alpha(v_0) = 1$ and $\alpha(v_1) = 0$. Thus, if $x = v_0, v_1$, $\mathtt{WHERE}(r_2)(\mathtt{value}_{\overline{\mathsf{wr}}}(t_x^{i-1}, x)) = 1$; so $(t_x^{i-1}, t_1) \not\in \overline{\mathsf{wr}}_x$. In conclusion, $w_x^{t_1} = t_0$; which implies (2) by definition of t_0.

- $\underline{t = t_2}$: As t_0, t_1 only write v_0, v_1 and v_2 we deduce that for every other key, $w_x^{t_2} = t_x^{i-1}$; which implies (2). Otherwise, we analyze the three subcases arising:
  - ∗ $\underline{x = v_2}$: In this case, t_0 does not write v_2; so there is only two options left, t_x^{i-1} and t_1. As (3) holds for round i, $\alpha(v_2) = 0$. Thus, as by induction hypothesis (2) holds for round $i-1$, $\texttt{value}_{\overline{\mathsf{wr}}}(t_{v_2}^{i-1}, v_2) = (v_2, 0)$ and hence, $\texttt{WHERE}(r_2)(\texttt{value}_{\overline{\mathsf{wr}}}(t_{v_2}^{i-1}, v_2)) = 1$. Therefore, $w_{v_2}^{t_2}$ must be t_1; which implies (2).
  - ∗ $\underline{x = v_0}$: Once again, there is only two possible options as t_1 does not write v_0. As (3) holds for round i, $\alpha(v_0) = 1$. Thus, as by induction hypothesis (2) holds for round $i-1$, $\texttt{value}_{\overline{\mathsf{wr}}}(t_{v_0}^{i-1}, v_0) = (v_0, 1)$ and hence, $\texttt{WHERE}(r_2)(\texttt{value}_{\overline{\mathsf{wr}}}(t_{v_0}^{i-1}, v_0)) = 1$. Therefore, $w_{v_0}^{t_2}$ must be t_0; which implies (2).
  - ∗ $\underline{x = v_1}$: We observe in this case that $\texttt{value}_{\overline{\mathsf{wr}}}(t_0, v_1) = (x, 1)$; so $\texttt{WHERE}(r_2)(\texttt{value}_{\overline{\mathsf{wr}}}(t_0, v_1)) = 1$. Therefore, there is only two possible options, t_1 and t_x^{i-1}. As $\overline{h}$ satisfies $\texttt{RC}$ and $(t_1, t_2) \in \overline{\mathsf{wr}}_{v_2}$, if $(t_x^{i-1}, t_2) \in \overline{\mathsf{wr}}_{v_1}$, we deduce that $(t_1, t_x^{i-1}) \in \mathsf{co}$. However, as $\texttt{round}(t_x^{i-1}) < \texttt{round}(t_1)$, $(t_x^{i-1}, t_1) \in \mathsf{wr}_{t_x^{i-1}}$; which is impossible as $\overline{\mathsf{wr}} \subseteq \mathsf{co}$. Thus, we conclude that $w_{v_2}^{t_2} = t_1$; which implies (2).

For proving (1) we observe that for every key $x \neq v_0, v_1, v_2$, $t_x^i = t_x^{i-1}$ and by induction hypothesis we conclude that $\texttt{value}_{\overline{\mathsf{wr}}}(t_x^i, x) = (x, \alpha(x))$. Moreover, as $(t_0, t_1) \in \mathsf{co}$ and $(t_1, t_2) \in \mathsf{co}$, $t_{v_0}^i = t_{v_2}^i = t_2$ and $t_{v_1}^i = t_1$. In addition, as (3) holds, $\alpha(v_0) = 1$ and $\alpha(v_1) = \alpha(v_2) = 0$. This allow us to conclude (1) also for the keys v_0, v_1 and v_2; so the inductive step is proven.

After proving (1), (2) and (3) we can conclude that φ is 1-in-3 satisfiable. For every round i, we observe that by (1) $\texttt{value}_{\overline{\mathsf{wr}}}(t_x^i, x) = (x, \alpha(x))$. Moreover, as (2) holds, $(t_x^i, t_0^i) \in \overline{\mathsf{wr}}_x$; where t_0^i is the first transaction in co among the transactions in round i. As $\overline{h}$ is a witness of h_φ, $\texttt{WHERE}(r_0^i)(\texttt{value}_{\overline{\mathsf{wr}}}(t_x^i, x)) = 0$; where r_0^i is the read event of t_0^i. Hence, exactly one variable among v_i^0, v_i^1 and v_i^2 has 1 as image by α. Therefore, φ is 1-in-3 satisfiable.

□

B.3 Proof of Theorem 4

Theorem 4. *Checking consistency of partial observation histories with bounded isolation configurations stronger than* $\texttt{RC}$ *is NP-complete.*

The structure of the proof is divided in two parts: proving that the problem is NP and proving that it is NP-hard. The fist part, corresponding to Lemma 18, is straightforward as, for any client history, we simply guess a suitable witness and a total order on its transactions for deducing its consistency applying Definition 6. The second part, corresponding to Lemmas 24 and 25 is more complicated. We use a novel reduction from 3-SAT. We encode a boolean formula φ in a history h_φ, s.t. h_φ is consistent iff φ is satisfiable. We first prove that the problem is indeed in NP (Lemma 18).

Lemma 18. *The problem of checking consistency for a client history with an isolation configuration stronger than* $\texttt{RC}$ *is in NP.*

Proof. Let $h = (T, \mathsf{so}, \mathsf{wr})$ a client history whose isolation configuration is stronger than $\texttt{RC}$. Guessing a witness of h, $\overline{h}$, and an execution of $\overline{h}$, $\xi = (\overline{h}, \mathsf{co})$, can be done in $\mathcal{O}(|\mathsf{Keys}| \cdot |\mathsf{events}(h)|^2 + |T|^2) \subseteq \mathcal{O}(|h|^3)$. By Lemma 5, checking if ξ is consistent is equivalent as checking if $\textsc{saturate}(\overline{h}, \mathsf{co})$ is an acyclic relation. As by Lemma 7, Algorithm 1 requires polynomial time, we conclude the result. □

For showing NP-hardness, we reduce 3-SAT to the problem of checking consistency of a partial observation history. Note that the problem is NP-hard in the case where the isolation configuration is not saturable, as discussed in Section 4.2, using the results in [7]. Therefore, we only prove it for the case where the isolation configuration is saturable.

Let $\varphi = \bigwedge_{i=1}^{n} C_i$ a CNF expression with at most 3 literals per clause (i.e. $C_i = l_i^1 \vee l_i^2 \vee l_i^3$). Without loss of generality we can assume that each clause contains exactly three literals and each literal in a clause refers to a different variable. We denote $\texttt{var}(l_i^j)$ to the variable that the literal l_i^j employs and $\mathsf{Vars}(\varphi)$ the set of all variables of φ.

We design a history h_φ with an arbitrary saturable isolation configuration encoding φ. Thus, checking φ-satisfiability would reduce to checking h_φ's consistency. Note that as $\mathsf{iso}(h_\varphi)$ is saturable, h_φ's consistency is equivalent to checking pco's acyclicity; where $\mathsf{pco} = \textsc{saturate}(h_\varphi, (\mathsf{so} \cup \mathsf{wr})^+)$. We use the latter characterization of consistency for encoding the formula φ in h_φ.

First of all, we consider every literal in φ independently. This means that even if two literals l_i^j and $l_{i'}^{j'}$ share its variable ($\texttt{var}(l_i^j) = \texttt{var}(l_{i'}^{j'})$) we will reason independently about them. For that, we employ keys $\texttt{var}(l_i^j)_i$ and $\texttt{var}(l_{i'}^{j'})_{i'}$. We later enforce additional constraints for ensuring that $\texttt{var}(l_i^j)_i$ and $\texttt{var}(l_{i'}^{j'})_{i'}$ coordinate so assignments on $\texttt{var}(l_i^j)_i$ coincide with assignments in $\texttt{var}(l_{i'}^{j'})_{i'}$. For simplicity in the explanation, whenever we talk about a literal l that is negated (for example $l := \neg x$), we denote by $\neg l$ to the literal x. Also, we use

```
INSERT($\{\mathtt{var}(l_i)^{+}_{(i,1)} : 1\}$)
...
INSERT($\{\mathtt{var}(l_i)^{+}_{(i,i-1)} : 1\}$)
INSERT($\{\mathtt{var}(l_i)^{+}_{(i,i+1)} : 1\}$)
...
INSERT($\{\mathtt{var}(l_i)^{+}_{(i,n)} : 1\}$)
INSERT($\{\mathtt{var}(l_i)^{-}_{(1,i)} : 1\}$)
...
INSERT($\{\mathtt{var}(l_i)^{-}_{(i-1,i)} : 1\}$)
INSERT($\{\mathtt{var}(l_i)^{-}_{(i+1,i)} : 1\}$)
...
INSERT($\{\mathtt{var}(l_i)^{-}_{(n,i)} : 1\}$)
INSERT($\{\mathtt{var}(c_i^j) : 1\}$)
DELETE($\lambda r : \mathsf{key}(r) = \mathtt{var}(l_i^j)_i$)
```

(a) Transaction t_i^j

```
INSERT($\{\mathtt{var}(l_i)^{-}_{(i,1)} : 1\}$)
...
INSERT($\{\mathtt{var}(l_i)^{-}_{(i,i-1)} : 1\}$)
INSERT($\{\mathtt{var}(l_i)^{-}_{(i,i+1)} : 1\}$)
...
INSERT($\{\mathtt{var}(l_i)^{-}_{(i,n)} : 1\}$)
INSERT($\{\mathtt{var}(l_i)^{+}_{(1,i)} : 1\}$)
...
INSERT($\{\mathtt{var}(l_i)^{+}_{(i-1,i)} : 1\}$)
INSERT($\{\mathtt{var}(l_i)^{+}_{(i+1,i)} : 1\}$)
...
INSERT($\{\mathtt{var}(l_i)^{+}_{(n,i)} : 1\}$)
INSERT($\{\mathtt{var}(c_i^j) : 1\}$)
INSERT($\{\mathtt{var}(c_i^{j-1 \bmod 3}) : 1\}$)
DELETE($\lambda r : \mathsf{key}(r) = \mathtt{var}(l_i^j)_i$)
```

(b) Transaction $\neg t_i^j$

```
INSERT($\{x \in \mathtt{A}_i^j : 0\}$)
SELECT($\lambda r : \mathsf{true}$)
```

(c) Transaction S_i^j

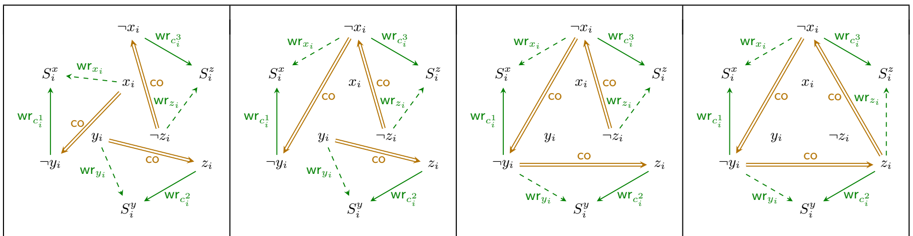
Fig. B.2: Description in full detail of the transactions t_i^j, $\neg t_i^j$ and S_i^j described in the proof of theorem 4 assuming $\mathtt{sign}(t_i^j) = +$; where $\mathtt{A}_i^j$ is the set of auxiliary variables for S_i^j. The case where $\mathtt{sign}(t_i^j) = -$ is analogous replacing in the first two instructions of both t_i^j and $\neg t_i^j$ $+$ by $-$ and vice versa.

indistinguishably x when referring to a variable in φ or to a homonymous key in h_φ. In addition, with the aim of simplifying the explanation, we assume hereinafter that any occurrence of indices i, i', j, j' satisfy that $1 \leq i, i' \leq n$ and $1 \leq j, j' \leq 3$.

For every clause $C_i = l_i^1 \vee l_i^2 \vee l_i^3$, we create nine transactions denoted by t_i^j, $\neg t_i^j$ and S_i^j. Figure B.2 shows in detail their definition, which we explain and justify during the following lines. The transaction t_i^j represents the state where l_i^j is satisfied while $\neg t_i^j$ represents the state where l_i^j is unsatisfied. Transaction S_i^j is in charge of selecting one of the two states. With this goal on mind, transactions t_i^j and $\neg t_i^j$ contain a DELETE event that deletes the key $\mathtt{var}(l_i^j)_i$ while S_i^j contains a SELECT event that does *not* read $\mathtt{var}(l_i^j)_i$ in h_φ. By Definition 9, any witnesses h' of h_φ must read $\mathtt{var}(l_i^j)_i$ from a transaction that deletes it. As h_φ contain only two transactions that deletes such key (t_i^j and $\neg t_i^j$), we can interpret that if S_i^j reads $\mathtt{var}(l_i^j)_i$ from t_i^j in h', then l_i^j is satisfied in φ while otherwise it is not.

For simplifying notation, as transactions $t_i^j, \neg t_i^j, S_i^j$ only have one read event, we define write-read dependencies directly from transactions instead of their read events. We also denote by $\mathtt{var}(t_i^j)$ and $\mathtt{var}(\neg t_i^j)$ to the variable $\mathtt{var}(l_i^j)$, associating each transaction with the variable of its associated literal.

We divide the construction of the history h_φ in two main parts. In the first part, we relate transactions $t_i^j, \neg t_i^j$ and S_i^j with the clause C_i, ensuring a satisfying valuation of clause C_i corresponds to a consistent history when restricted to its associated transactions. In the second part, we link transactions associated to different clauses (for example t_i^j with $t_{i'}^{j'}$, $i \neq i'$), for ensuring that valuations



(a) No co-cycle when all literals are satisfied

(b) No co-cycle when two literals are satisfied

(c) No co-cycle when only one literal is satisfied

(d) One co-cycle appears when no literal is satisfied

Fig. B.3: Transformation of the clause $C_i = x_i \vee y_i \vee \neg z_i$ into part of the history. Solid wr-edges in h_φ represent the constraints of the clause while dashed wr-edges, belonging only to the witnesses of h_φ, reflect the literals satisfied.

are consistent between clauses (i.e. a variable is not assigned 1 in clause C_i and 0 in clause $C_{i'}$).

For the first part of h_φ's construction, we observe that "at least one literal among l_i^1, l_i^2 or l_i^3 must be satisfied" is equivalent to "$\neg l_i^1$, $\neg l_i^2$ and $\neg l_i^3$ cannot be satisfied at the same time". Thus, we add write-read dependencies to the history in such a way that if the three values that do not satisfy the clause are read by a witness h' of h_φ, axiom Read Committed forces h' to be inconsistent. We use an auxiliary key c_i^j written by transactions t_i^j, $\neg t_i^j$ and $\neg t_i^{(j+1) \bmod 3}$ and read by transaction S_i^j; enforcing $(\neg t_i^{(j+1) \bmod 3}, S_i^j) \in \mathsf{wr}_{c_i^j}$. Thanks to variable c_i^j, if $(\neg t_i^j, S_i^j) \in \mathsf{wr}_{\mathtt{var}(l_i^j)_i}$ in such witness h', for any consistent execution of h' with commit order co, $(\neg t_i^j, \neg t_i^{j+1 \bmod 3}) \in \mathsf{co}$. Hence, if h' is consistent, for every i there must exist a j s.t.$(t_i^j, S_i^j) \in \mathsf{wr}_{\mathtt{var}(l_i^j)_i}$. Otherwise, every commit order witnessing h''s consistency would be cyclic; which is a contradiction.

In Figure B.3 we see how such co-cycle arise on any commit order witnessing h_φ's consistency; where φ contains the clause $C_i = x_i \vee y_i \vee \neg z_i$.

$$\mathtt{sign}(t) = \begin{cases} + \text{ if } t = t_i^j \wedge l_i^j = \mathtt{var}(l_i^j) \\ - \text{ if } t = t_i^j \wedge l_i^j = \neg\mathtt{var}(l_i^j) \\ - \text{ if } t = \neg t_i^j \wedge l_i^j = \mathtt{var}(l_i^j) \\ + \text{ if } t = \neg t_i^j \wedge l_i^j = \neg\mathtt{var}(l_i^j) \\ \perp \text{ otherwise} \end{cases} \quad \mathtt{opsign}(t) = \begin{cases} + \text{ if } \mathtt{sign}(t) = - \\ - \text{ if } \mathtt{sign}(t) = + \\ \perp \text{ otherwise} \end{cases} \tag{9}$$

For the second part of h_φ's construction, we utilize the functions sign and opsign described in Equation 9. The function sign describes when a literal l_i^j is *positive* (i.e. $l_i^j = \mathtt{var}(l_i^j)$) or *negative* (i.e. $l_i^j = \neg\mathtt{var}(l_i^j)$). If l_i^j is positive, it assigns to transaction t_i^j the symbol + and to $\neg t_i^j$ the symbol −; while if l_i^j is negative, the opposite. Such symbol is denoted the *sign* of a transaction. Hence, for each transaction t_i s.t. $\mathtt{sign}(t_i) \neq \perp$ (i.e. t_i is either t_i^j or $\neg t_i^j$), we introduce

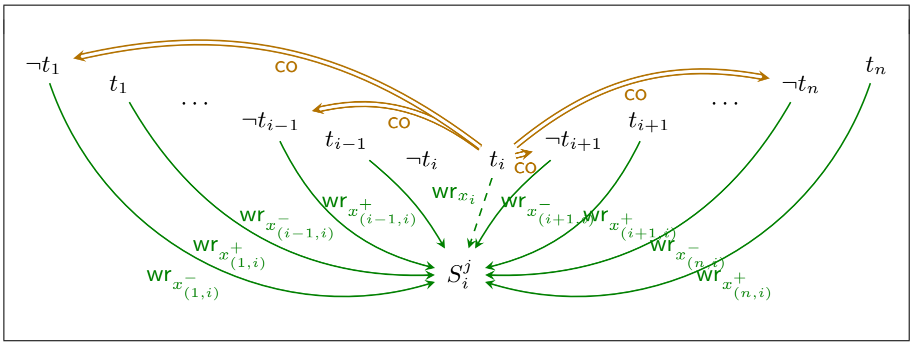

$\neg t_1$
t_1
$\dots$
$\neg t_{i-1}$
t_{i-1}
$\neg t_i$
t_i
$\neg t_{i+1}$
t_{i+1}
$\dots$
$\neg t_n$
t_n
co
co
co
co
wr_{x_i}
$\mathsf{wr}_{x^-_{(i-1,i)}}$
$\mathsf{wr}_{x^+_{(i-1,i)}}$
$\mathsf{wr}_{x^+_{(1,i)}}$
$\mathsf{wr}_{x^-_{(1,i)}}$
$\mathsf{wr}_{x^-_{(i+1,i)}}$
$\mathsf{wr}_{x^+_{(i+1,i)}}$
$\mathsf{wr}_{x^-_{(n,i)}}$
$\mathsf{wr}_{x^+_{(n,i)}}$
S_i^j

Fig. B.4: Commit edges between transactions of different sign associated to variable $x = \texttt{var}(l_i^j)$. Superindices j are omitted for legibility. For simplicity on the Figure, we assume that $\texttt{sign}(t_k) = +$ and $\texttt{sign}(\neg t_k) = -$; the situation generalizes for any other setting. If S_i^j would read x_i from t_i in a witness h' of h_φ (respectively $\neg t_i$), for every $i' \neq i$ $(t_i, \neg t_{i'}) \in \mathsf{co}$, (resp. $(\neg t_i, t_{i'}) \in \mathsf{co}$).

$n-1$ `INSERT` events, one per key $\texttt{var}(l_i^j)^{\texttt{sign}(t_i)}_{(i,i')}$, $i' \neq i$, that write on that exact key. Those `INSERT` events are read by transaction $S_{i'}^{j'}$ (i.e. $(t_i, S_{i'}^{j'}) \in \mathsf{wr}_x$, where $x = \texttt{var}(l_i^j)^{\texttt{sign}(t_i)}_{(i,i')}$). In addition, we introduce in t_i another $n-1$ `INSERT` events that writes the key $\texttt{var}(t_i)^{\texttt{opsign}(t_i)}_{(i',i)}$. Figure B.2 describe the full description of transactions t_i^j and $\neg t_i^j$.

The auxiliary keys $\texttt{var}(l_i^j)^{\texttt{sign}(t_i)}_{(i,i')}$ and $\texttt{var}(l_i^j)^{\texttt{opsign}(t_i)}_{(i,i')}$ are key to ensure consistency between clauses. Intuitively, if S_i^j reads $\texttt{var}(l_i^j)_i$ from a positive transaction t in a consistent execution of h_φ, $\xi = (\overline{h}, \mathsf{co})$, and t' is a negative transaction s.t. $\texttt{var}(t) = \texttt{var}(t')$, then $(t, t') \in \mathsf{co}$; where $\overline{h}$ is a witness of h_φ. Hence, any other transaction $S_{i'}^{j'}$ must read $\texttt{var}(l_{i'}^{j'})_{i'}$ also from a positive transaction in $\overline{h}$; otherwise co would be cyclic, which is impossible as co must be a total order. This phenomenon, that is depicted in Figure B.4, ensures that $\texttt{var}(l_i^j)$ is always read from transactions with the same sign. In conclusion, we can establish consistent valuation of the variables of φ based on the write-read dependencies of the witnesses of h_φ.

We introduce a succint final part on the construction of h_φ for technical reasons. Indeed, any witness of h_φ ensures that wr_x^{-1} is a total function for any $x \in \mathsf{Keys}$. We impose a few additional constraints on h_φ so we can better characterize the witnesses of h_φ. First, we assume that there exists an initial transaction that inserts, for every key x, a dummy value different from $\dagger_x$ (for example 0). Then, we impose that t_i^j and $\neg t_i^j$ read every key x from the initial transaction. Finally, for the case of transactions S_i^j, we define the set of *auxiliary keys* $\mathtt{A}_i^j$ that contain every key different from $c_i^j, \texttt{var}(l_i^j)_i, \texttt{var}(l_i^j)^+_{(i',i)}$ and $\texttt{var}(l_i^j)^-_{(i',i)}$. We introduce on S_i^j an `INSERT` event that writes every key in $\mathtt{A}_i^j$

with an abritrary value (for example, 0). Hence, S_i^j reads every key in A_i^j from its own insert and no extra write-read dependency is required.

With this technical addendum, we define $h_\varphi = (T, \mathsf{so}, \mathsf{wr})$ as the conjunction of all transactions and relations described above. In such case, the only information missing in h_φ to be a full history is $\mathsf{wr}^{-1}_{\mathtt{var}(l_i^j)_i}(S_i^j)$. We assume that no more variables than the ones aforementioned belong to Keys.

The proof of NP-hardness goes as follows: first, we prove in Lemma 19 that h_φ is indeed a polynomial transformation of φ. Then, as $\mathsf{iso}(h_\varphi)$ is saturable, by Theorem 2 we observe that it suffices to prove that φ is satisfiable if and only there is a witness $\overline{h}$ of h_φ s.t. the relation $\mathsf{pco}_{\overline{h}} = \textsc{saturate}(\overline{h}, (\mathsf{so} \cup \overline{\mathsf{wr}})^+)$ is acyclic. For simplifying the reasoning when $\mathsf{iso}(h_\varphi)$ has an arbitrary isolation configuration, we rely on Lemma 20 for reducing the proof at the case when $\mathsf{iso}(h_\varphi) = \mathtt{RC}$.

Hence, we prove on Lemma 24 that on one hand, whenever φ is satisfiable we can construct a witness $\overline{h}$ of h_φ based on such assignment for which $\mathsf{pco}_{\overline{h}}$ is acyclic. For that, we require Lemmas 21 to 23. On the other hand, whenever there is a consistent witness $\overline{h}$ of h_φ, we prove on Lemma 25 that we can construct a satisfying assignment of φ based on the write-read dependencies in $\overline{h}$. In this case, we require once more Lemma 21.

Lemma 19. *The history h_φ has polynomial size on the length of φ.*

Proof. Let φ a CNF with n clauses and 3 literals per clause. Then, as φ has $3n$ literals, h_φ employs $9n$ transactions plus one additional one ($\mathtt{init}$). The number of keys, $|\mathsf{Keys}|$, is quadratic on n as transactions t_i^j and $\neg t_i^j$ insert $\mathcal{O}(n)$ keys while S_i^j only insert keys also inserted by other transactions. Moreover, the number of events in h_φ, $\mathsf{events}(h_\varphi)$ is in $\mathcal{O}(|\mathsf{Keys}|) = \mathcal{O}(n^2)$ as transactions $t_i^j, \neg t_i^j$ have one $\mathtt{INSERT}$ event per keys inserted (and they insert $\mathcal{O}(n)$ keys) and one $\mathtt{DELETE}$ event and transactions S_i^j only have two events. In addition, $\mathsf{so} \subseteq T \times T$ and $\mathsf{wr} \subseteq \mathsf{Keys} \times \mathsf{events}(h_\varphi) \times \mathsf{events}(h_\varphi)$; so their size is also polynomial on n. Thus, h_φ is a polynomial transformation of φ. ◻

One caveat of h_φ is that its isolation configuration is unknown. Lemma 20 express that, in the particular case of h_φ, all saturable isolation levels stronger than $\mathtt{RC}$ are equivalent (they impose the same constraints). Hence, thereinafter we can assume without loss of generality that $\mathsf{iso}(h_\varphi) = \mathtt{RC}$.

Lemma 20. *Under history h_φ, $\mathsf{iso}(h_\varphi)$ is equivalent to $\mathtt{RC}$ (i.e. $\mathsf{iso}(h_\varphi)$ is both weaker and stronger than $\mathtt{RC}$.)*

Proof. Let $\overline{h} = (T, \mathsf{so}, \overline{\mathsf{wr}})$ be any witness of h_φ and let $h^{\mathtt{RC}}$ be the history that only differ with $\overline{h}$ on its isolation configuration ($\mathsf{iso}(h^{\mathtt{RC}}) = \mathtt{RC}$ instead of $\mathsf{iso}(h)$). We prove that $\overline{h}$ and $h^{\mathtt{RC}}$ are simultaneously consistent or inconsistent.

As both $\mathsf{iso}(h_\varphi)$ and $\mathtt{RC}$ are saturable, by Theorem 2, the proof is equivalent to prove that $\mathsf{pco}_{\overline{h}}$ and $\mathsf{pco}_{\mathtt{RC}}$ are simultaneously cyclic or acyclic; where $\mathsf{pco}_{\overline{h}} = \textsc{saturate}(\overline{h}, (\mathsf{so} \cup \overline{\mathsf{wr}})^+)$ and $\mathsf{pco}_{\mathtt{RC}} = \textsc{saturate}(h^{\mathtt{RC}}, (\mathsf{so} \cup \overline{\mathsf{wr}})^+)$. We prove that the two relations coincide, which allow us to conclude the result.

We observe that as $\mathsf{iso}(\overline{h})$ is weaker than $\mathtt{RC}$, $\mathsf{pco}_{\mathtt{RC}} \subseteq \mathsf{pco}_{\overline{h}}$. Thus, it suffices to prove that $\mathsf{pco}_{\overline{h}} \subseteq \mathsf{pco}_{\mathtt{RC}}$. Let t, t' be two transactions s.t. $(t, t') \in \mathsf{pco}_{\overline{h}}$ and let us prove that $(t, t') \in \mathsf{pco}_{\mathtt{RC}}$. As $(\mathsf{so} \cup \overline{\mathsf{wr}})^+ \subseteq \mathsf{pco}_{\mathtt{RC}}$; we can assume without loss of generality that $(t, t') \in \mathsf{pco}_{\overline{h}} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$. In such case, there exists $r \in \mathsf{reads}(\overline{h}), x \in \mathsf{Keys}$ and $v \in \mathsf{vis}(\mathsf{iso}(\overline{h})(\mathsf{tr}(r)))$ s.t. $t' = \overline{\mathsf{wr}}_x{}^{-1}(r)$ and $\mathsf{v}(\mathsf{pco}_{\overline{h}})(t, r, x)$ holds in $\overline{h}$. As $\mathsf{iso}(\overline{h})$ is saturable, $(t, r) \in (\mathsf{so} \cup \overline{\mathsf{wr}})^+$.

First, we note that $\mathsf{tr}(r) \neq \mathtt{init}$ as it does not contain any read event. As t' is a $(\mathsf{so} \cup \overline{\mathsf{wr}})^+$-predecessor of $\mathsf{tr}(r)$, and transactions S_i^j are $(\mathsf{so} \cup \overline{\mathsf{wr}})$-maximal, t' is not a S_i^j transaction; so it must be a t_i^j transaction. However, note that by construction of transactions t_i^j, their only $(\mathsf{so} \cup \overline{\mathsf{wr}})$-predecessor is $\mathtt{init}$. Thus, their only $(\mathsf{so} \cup \overline{\mathsf{wr}})$-succesors can be transactions $S_{i'}^{j'}$; transactions that do not have $(\mathsf{so} \cup \overline{\mathsf{wr}})$-successors. In conclusion, if $(t', \mathsf{tr}(r)) \in (\mathsf{so} \cup \overline{\mathsf{wr}})^+$, $(t', \mathsf{tr}(r)) \in \mathsf{so} \cup \overline{\mathsf{wr}}$, and therefore, $(t', r) \in (\mathsf{so} \cup \overline{\mathsf{wr}}); \mathsf{po}^*$. □

Lemma 21 states a characterization of all commit order cycles imposed by the axiom $\mathtt{RC}$ that only relate the nine transactions associated to a clause in φ.

Lemma 21. *Let $\overline{h} = (T, \mathsf{so}, \overline{\mathsf{wr}})$ a witness of h_φ. For a fixed i, there is a $\mathsf{pco}_{\overline{h}}$-cycle relating $\mathtt{init}$, $t_i^j, \neg t_i^j$ and S_i^j, $1 \leq j \leq 3$ in h if and only if for all $1 \leq j \leq 3$, $(\neg t_i^j, S_i^j) \in \overline{\mathsf{wr}}_{\mathtt{var}(l_i^j)}$ in $\overline{h}$.*

Proof. A graphical description of the different cases of this proof can be seen in Figure B.3.

$\underline{\Longleftarrow}$

Let us suppose that for every j, $1 \leq j \leq 3$, $(\neg t_i^j, S_i^j) \in \overline{\mathsf{wr}}_{\mathtt{var}(l_i^j)_i}$. As $\neg t_i^j$ writes c_i^j and $(\neg t_i^{(j+1) \bmod 3}, S_i^j) \in \overline{\mathsf{wr}}_{c_i^j}$, by axiom $\mathtt{RC}$ we deduce that, $(\neg t_i^j, \neg t_i^{(j+1) \bmod 3}) \in \mathsf{pco}_{\overline{h}}$. Therefore, there is a $\mathsf{pco}_{\overline{h}}$-cycle between transactions $\neg t_i^1, \neg t_i^2$ and $\neg t_i^3$.

$\underline{\Longrightarrow}$

First, note that $\mathsf{so} \cup \overline{\mathsf{wr}}$ is acyclic, so any $\mathsf{pco}_{\overline{h}}$-cycle has to include at least one $\mathsf{pco}_{\overline{h}} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$-dependency. Hence, let t, t' be distinct transactions such that $(t', t) \in \mathsf{pco}_{\overline{h}} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$ is an edge belonging to such cycle. By axiom $\mathsf{Read\ Committed}$, this implies that there exists a read event r and a key x s.t. $(t, r) \in \overline{\mathsf{wr}}_x$ and $(t, r) \in (\mathsf{so} \cup \overline{\mathsf{wr}}); \mathsf{po}^*$. Note that in particular this means that t' and t are two distinct $(\mathsf{so} \cup \overline{\mathsf{wr}})$-succesors of $\mathsf{tr}(r)$.

We observe that $\mathsf{tr}(r) \neq \mathtt{init}$ as $\mathtt{init}$ does not contain any read event. Moreover, $\mathsf{tr}(r) \neq t_i^j, \neg t_i^j$ as those transactions have only one $(\mathsf{so} \cup \overline{\mathsf{wr}})$-predecessor, $\mathtt{init}$. Hence, there exists j s.t. $\mathsf{tr}(r) = S_i^j$. In this case, every ke written by t_i^j or $\neg t_i^j$ besides c_i^j and $\mathtt{var}(l_i^j)_i$ is read by S_i^j from the $\mathtt{INSERT}$ event in its own transaction. We distinguish between two cases:

- $\underline{x = \mathtt{var}(l_i^j)_i}$: The only transactions that write $\mathtt{var}(l_i^j)_i$ are t_i^j, $\neg t_i^j$, $\mathtt{init}$ and transactions $S_{i'}^{j'}$. However, transactions $S_{i'}^{j'}$ have only one $(\mathsf{so} \cup \overline{\mathsf{wr}})$-succesor, $\mathtt{init}$ in h_φ. As $\forall x \neq \mathtt{var}(l_i^j)_i, \overline{\mathsf{wr}}_x{}^{-1}(S_i^j) = \mathsf{wr}_x{}^{-1}(S_i^j)$, one of them, $\texttt{init}$ must be either t or t'. However, $t \neq \texttt{init}$ as $\texttt{init}$ does not delete $\texttt{var}(l_i^j)_i$; so $t = \texttt{init}$. But in such case, $(t', t) \in \mathsf{so}$; which contradicts that $(t', t) \in \mathsf{pco}_{\overline{h}} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$. This proves that this case is impossible.

- $\underline{x = c_i^j}$: In such case, as $(\neg t_i^{j+1 \text{ mod } 3}, S_i^j) \in \mathsf{wr}_{c_i^j}$, $t = \neg t_i^{j+1 \text{ mod } 3}$. The only transactions that writes c_i^j and are $(\mathsf{so} \cup \overline{\mathsf{wr}})$-predecessors of S_i^j are $\texttt{init}, t_i^j$ and $\neg t_i^j$. As $(\texttt{init}, t) \in \mathsf{so}$; $t \neq \texttt{init}$. Thus, any of the other two transactions are candidates to be t'. Note that $(t', t) \in \mathsf{pco}_{\overline{h}}$ is part of a cycle; so let t'' be a transaction s.t. $(t'', t') \in \mathsf{pco}_{\overline{h}}$.
If $(t'', t') \in (\mathsf{so} \cup \overline{\mathsf{wr}})$ would hold, as for every key x, $\overline{\mathsf{wr}}_x{}^{-1}(t) = \mathsf{wr}_x{}^{-1}(t)$, $t'' = \texttt{init}$. As (t', t) is part of a $\mathsf{pco}_{\overline{h}}$ cycle and $t \neq \texttt{init}$, there must exist a transaction $t''' \neq t'' = \texttt{init}$ s.t. $(t''', t'') \in \mathsf{pco}_{\overline{h}}$ is part of such cycle. Note that $(t''', \texttt{init}) \in \mathsf{pco}_{\overline{h}} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})$ by construction of h_φ. Hence, there exists a key y and a read event r' s.t. $(\texttt{init}, r') \in \overline{\mathsf{wr}}_y$ and $(t''', r') \in (\mathsf{so} \cup \overline{\mathsf{wr}}); \mathsf{po}^*$. By construction of h_φ, if $(\texttt{init}, r') \in \overline{\mathsf{wr}}_y$ then $\mathsf{tr}(r')$ must be $t_i^{j'}$ for some j'. But as we mentioned earlier, such transactions only have one $(\mathsf{so} \cup \overline{\mathsf{wr}})$-predecessor, $\texttt{init}$; so it is impossible that $(t'', t') \in (\mathsf{so} \cup \overline{\mathsf{wr}})$.
Hence, $(t'', t') \in \mathsf{pco}_{\overline{h}} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})$. Replicating the same argument as before we can deduce that there exists a j' s.t. $(t'', S_i^{j'}) \in \overline{\mathsf{wr}}_{c_i^{j'}}$, $t' = \neg t_i^{j'+1 \text{ mod } 3}$ and t'' is either $t_i^{j'}$ or $\neg t_i^{j'}$. However, as discussed before, t' could only be $\neg t_i^j$ or t_i^j. Therefore, $t' = \neg t_i^j$ and $j = j' + 1 \text{ mod } 3$.
Finally, as $t'' \neq t$, there must exist a transaction t''' s.t. $(t''', t'') \in \mathsf{pco}_{\overline{h}}$. By the same argument once more, there exists an index j'' s.t. $t'' = \neg t_i^{j''+1 \text{ mod } 3}$, $(t''', S_i^{j''}) \in \overline{\mathsf{wr}}_{c_i^{j''}}$ and t''' is either $t_i^{j''}$ or $\neg t_i^{j''}$. Once more, as t'' could only be $\neg t_i^{j'}$ or $t_i^{j'}$, we deduce that $j' = j'' + 1 \text{ mod } 3$ and $t'' = \neg t_i^{j'}$. Note that in this case $j = j'' + 2 \text{ mod } 3$. Thus, $t = \neg t_i^{j+1 \text{ mod } 3} = \neg t_i^{j''} = t'''$. In conclusion, if such cycle exists it contain exactly the transactions $\neg t_i^1, \neg t_i^2$ and $\neg t_i^3$ and for each of them, $(\neg t_i^j, S_i^j) \in \overline{\mathsf{wr}}_{\texttt{var}(l_i^j)_i}$.

□

Lemma 22 states that any $\mathsf{pco}_{\overline{h}}$-dependencies imposed by the axiom $\texttt{RC}$ on transactions t, t' associated to diferent clauses in φ are related to valuation choices of literals in φ.

Lemma 22. *Let $\overline{h} = (T, \mathsf{so}, \overline{\mathsf{wr}})$ a witness of the history h_φ. For every pair of transactions t, t' and indices i, j, if $\texttt{var}(t) = \texttt{var}(t')$, t' $\mathsf{deletes}$ $\texttt{var}(l_i^j)_i$, $t \neq \neg t_i^{j+1 \text{ mod } 3}$ and $(t', t) \in \mathsf{pco}_{\overline{h}} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$ in h, then $(t', S_i^j) \in \overline{\mathsf{wr}}_{\texttt{var}(l_i^j)_i}$.*

Proof. Let i, j be indices and t, t' be distinct transactions such that $t \neq \neg t_i^{j+1 \text{ mod } 3}$ and $(t', t) \in \mathsf{pco}_{\overline{h}} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$. Hence, $(t', t) \in \mathsf{pco}_{\overline{h}} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$, by axiom $\mathsf{Read\ Committed}$, there must exist a key x and a read event r s.t. $(t, r) \in \overline{\mathsf{wr}}_x$, t' writes x and $(t', r) \in (\mathsf{so} \cup \overline{\mathsf{wr}}); \mathsf{po}^*$. We characterize the possible candidates of transactions $t, t', \mathsf{tr}(r)$ and key x.

First, as $\mathsf{tr}(r)$ has two different $(\mathsf{so} \cup \overline{\mathsf{wr}})$-predecessors, $\mathsf{tr}(r) \neq \texttt{init}, t_{i'}^{j'}$; for any indices i', j'. Hence, there must exist indices i', j' s.t. $\mathsf{tr}(r) = S_{i'}^{j'}$.

Next we deduce that t and t' belongs to different clauses. As t' $\mathsf{deletes}$ $\texttt{var}(l_i^j)_i$, we deduce that t' is either t_i^j or $\neg t_i^j$. Hence, as $t \neq \neg t_i^{j+1 \bmod 3}$, neither t_i^j nor $\neg t_i^j$ are $(\mathsf{so} \cup \overline{\mathsf{wr}})$-predecessors of S_i^j but both t and t' are $(\mathsf{so} \cup \overline{\mathsf{wr}})$-predecessors of S_i^j, we then deduce that t and t' belong to different clauses.

Finally, we deduce that $i' = i$ and $(t', S_i^j) \in \overline{\mathsf{wr}}_{\texttt{var}(l_i^j)_i}$. As x is written by t and t' and $x \notin \texttt{A}_{i'}^{j'}$; either t or t' are associated to the same clause as $S_{i'}^{j'}$. If t would be associated to clause $C_{i'}$, then t should be either $t_{i'}^{j'}$ or $\neg t_{i'}^{j'}$ and $x = \texttt{var}(l_{i'}^{j'})_{i'}$. However, this contradicts that t' writes $\texttt{var}(l_{i'}^{j'})_{i'}$ as t' is either t_i^j or $\neg t_i^j$. Hence, as t is not associated to clause $C_{i'}$, $i' = i$. As $(t', S_i^j) \notin (\mathsf{so} \cup \mathsf{wr})$ but $(t', S_i^j) \in (\mathsf{so} \cup \overline{\mathsf{wr}})$ and $\overline{\mathsf{wr}}_y = \mathsf{wr}_y$ for any key $y \neq \texttt{var}(l_i^j)_i$, we conclude that $(t', S_i^j) \in \overline{\mathsf{wr}}_{\texttt{var}(l_i^j)_i}$. □

Lemma 23 states that $\mathsf{pco}_{\overline{h}}$ does not contain tuples of transactions associated to literals with equal variable and sign.

Lemma 23. *Let $\overline{h} = (T, \mathsf{so}, \overline{\mathsf{wr}})$ a witness of the history h_φ. For every pair of transactions t, t' and indices i, j, if $\texttt{sign}(t) = \texttt{sign}(t')$, $\texttt{var}(t) = \texttt{var}(t') = \texttt{var}(l_i^j)$, $(t, S_i^j) \in \overline{\mathsf{wr}}_{\texttt{var}(l_i^j)_i}$ and $t' \neq \neg t_i^{j-1 \bmod 3}$ then $(t', t) \notin \mathsf{pco}_{\overline{h}} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$.*

Proof. We reason by contradiction. Let us suppose that t, t' are a pair of transactions such that $\texttt{sign}(t) = \texttt{sign}(t')$, $\texttt{var}(t) = \texttt{var}(t') = \texttt{var}(l_i^j)$, $(t, S_i^j) \in \overline{\mathsf{wr}}_{\texttt{var}(t)_i}$, $t' \neq \neg t^{j-1 \bmod 3}$ and $(t', t) \in \mathsf{pco}_{\overline{h}} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$, for some indices i, j. As $(t, S_i^j) \in \overline{\mathsf{wr}}_{\texttt{var}(l_i^j)_i}$, t is either t_i^j or $\neg t_i^j$. Moreover, as $(t', t) \in \mathsf{pco}_{\overline{h}} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$, by axiom $\mathsf{Read\ Committed}$ we deduce that there exists a key x and a read event r s.t. $(t', r) \in (\mathsf{so} \cup \overline{\mathsf{wr}}); \mathsf{po}^*$, $(t, r) \in \overline{\mathsf{wr}}_x$ and t' writes x.

We first prove that t' and t are associated to different clauses. As $(\texttt{init}, t) \in \mathsf{so}$, $t' \neq \texttt{init}$. Next, as $(t', r) \in (\mathsf{so} \cup \overline{\mathsf{wr}}); \mathsf{po}^*$ and transactions $S_{i'}^{j'}$ are $\mathsf{so} \cup \overline{\mathsf{wr}}$-maximal, we deduce that there must exist a pair of indices i', j' s.t. $t' = t_{i'}^{j'}$ or $\neg t_{i'}^{j'}$. Moreover, as $t' \neq \neg t_i^{j-1 \bmod 3}$, t is either t_i^j or $\neg t_i^j$. In addition, as in any witness of h_φ both t_i^j and $\neg t_i^j$ cannot be $(\mathsf{so} \cup \overline{\mathsf{wr}})$-predecessors of S_i^j, we deduce that $i' \neq i$.

Finally we contradict the hypothesis proving that $\texttt{sign}(t) \neq \texttt{sign}(t')$. If $i' \neq i$, $t \neq \texttt{init}$ but t is a $\overline{\mathsf{wr}}$-predecessor of $\mathsf{tr}(r)$, there must exist indices i'', j'' s.t. $\mathsf{tr}(r) = S_{i''}^{j''}$. Hence, as $x \in \texttt{A}_{i''}^{j''}$ and it is written by t and t', i'' must be either i' or i. However, $i'' \neq i$ as in that case, $x = \texttt{var}(l_i^j)_i$ and t' does not write $\texttt{var}(l_i^j)_i$. Hence, $i'' = i' \neq i$ and $x = \texttt{var}(l_i^j)_{(i',i)}^{\texttt{sign}(t')}$. However, as t writes x, by construction of h_φ, we must conclude that $\texttt{sign}(t) \neq \texttt{sign}(t')$. Thus, as we reached a contradiction, the lemma holds. □

Lemma 24. *For every boolean formula φ, if φ is satisfiable then there is a consistent witness $\overline{h}$ of h_φ.*

Proof. Let $\alpha : \mathsf{Vars}(\varphi) \to \{0,1\}$ an assignment that satisfies φ. Let $h^\alpha_\varphi = (T, \mathsf{so}, \overline{\mathsf{wr}})$ the extension of h_φ s.t. for every i, j, $(t^j_i, S^j_i) \in \overline{\mathsf{wr}}_{\mathtt{var}(l^j_i)_i}$ if $l^j_i[\alpha(\mathtt{var}(l^j_i))/\mathtt{var}(l^j_i)] = \mathsf{true}$ and $(\neg t^j_i, S^j_i) \in \overline{\mathsf{wr}}_{\mathtt{var}(l^j_i)_i}$ otherwise. Note that for every two transactions t, t' s.t. $\mathtt{var}(t) = \mathtt{var}(t')$, $\alpha(\mathtt{var}(t)) = \alpha(\mathtt{var}(t'))$. Hence, if $(t, S^j_i) \in \overline{\mathsf{wr}}_{\mathtt{var}(t)_i}$ and $(t', S^{j'}_{i'}) \in \overline{\mathsf{wr}}_{\mathtt{var}(t')_{i'}}$ then $\mathtt{sign}(t) = \mathtt{sign}(t')$. In addition, by construction of h_φ, for every transaction S^j_i, the only variable x such that $\mathsf{wr}_x^{-1}(S^j_i)\uparrow$ is $x = \mathtt{var}(l^j_i)$. Thus, for every $x \in \mathsf{Keys}$, $\overline{\mathsf{wr}}_x{}^{-1}$ is defined for any read that does not read locally and therefore, h^α_φ is a full history that extends h_φ.

Let us prove that h^α_φ is consistent. As mentioned before, thanks to Theorem 2, we can reduce the problem of checking if h^α_φ is consistent to the problem of checking if $\mathsf{pco}_{h^\alpha_\varphi} = \textsc{saturate}(h^\alpha_\varphi, (\mathsf{so} \cup \overline{\mathsf{wr}})^+)$ is acyclic.

We reason by contradiction, assuming there is a $\mathsf{pco}_{h^\alpha_\varphi}$-cycle and reaching a contradiction. Clearly $\mathsf{so} \cup \overline{\mathsf{wr}}$ is acyclic as $\mathsf{so} \cup \mathsf{wr}$ is acyclic, transactions S^j_i are $(\mathsf{so} \cup \overline{\mathsf{wr}})$-maximal and $\overline{\mathsf{wr}} \setminus \mathsf{wr}$ only contains tuples (t^j_i, S^j_i) or $(\neg t^j_i, S^j_i)$. Thus, any $\mathsf{pco}_{h^\alpha_\varphi}$-cycle in h^α_φ contains at least one edge $(t', t) \in \mathsf{pco}_{h^\alpha_\varphi} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$; so let be t, t' such a pair of distinct transactions s.t. $(t', t) \in \mathsf{pco}_{h^\alpha_\varphi} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$ and (t', t) is part of the $\mathsf{pco}_{h^\alpha_\varphi}$-cycle.

First, we observe that by construction of h_φ, transactions S^j_i are $(\mathsf{so} \cup \overline{\mathsf{wr}})$-maximal. Moreover, they also $\mathsf{pco}_{h^\alpha_\varphi}$-maximal: by contradiction, if there was a transaction u^j_i s.t. $(S^j_i, u^j_i) \in \mathsf{pco}_{h^\alpha_\varphi} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})+$, by axiom $\mathsf{Read\ Committed}$, there would be a variable a read event r s.t. $(S^j_i, r) \in (\mathsf{so} \cup \overline{\mathsf{wr}}); \mathsf{po}^*$; which is impossible. We also observe that $\mathtt{init}$ is not only $(\mathsf{so} \cup \overline{\mathsf{wr}})$-minimal but $\mathsf{pco}_{h^\alpha_\varphi}$-minimal. By the same argument, if there would be a transaction $u \neq \mathtt{init}$ s.t. $(u, \mathtt{init}) \in \mathsf{pco}_{h^\alpha_\varphi} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$, by axiom $\mathsf{Read\ Committed}$, there should be a read event r and a key x s.t. $(\mathtt{init}, r) \in \overline{\mathsf{wr}}_x$ and $(u', r) \in (\mathsf{so} \cup \overline{\mathsf{wr}}); \mathsf{po}^*$. However, by construction of h_φ, the only transactions that read a variable from $\mathtt{init}$ are t^j_i; transactions with only one $(\mathsf{so} \cup \overline{\mathsf{wr}})$-predecessor. This shows that such transaction u does not exist. Altogether, the $\mathsf{pco}_{h^\alpha_\varphi}$-cycle can only contain pairs of transactions t^j_i and $\neg t^j_i$. In particular, as transactions t^j_i have only one $(\mathsf{so} \cup \overline{\mathsf{wr}})$-predecessor, $\mathtt{init}$, such $\mathsf{pco}_{h^\alpha_\varphi}$-cycle is in $\mathsf{pco}_{h^\alpha_\varphi} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$.

Next, we note that, as every clause C_i is satisfied by α, there exists an index j s.t. $(t^j_i, S^j_i) \in \mathtt{var}(l^j_i)$. By Lemma 21, we know there is no $\mathsf{pco}_{h^\alpha_\varphi}$-cycle relating the nine transactions associated with clause C_i and $\mathtt{init}$. Therefore, a $\mathsf{pco}_{h^\alpha_\varphi}$-cycle has to involve at least two transactions from different clauses. Hence, we can assume without loss of generality that t and t' belong to the same clause.

As $(t', t) \in \mathsf{pco}_{h^\alpha_\varphi} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$, there must exist a key x and a read event r_x s.t. t writes x, $(t, r_x) \in \overline{\mathsf{wr}}_x$ and $(t', r) \in (\mathsf{so} \cup \overline{\mathsf{wr}}); \mathsf{po}^*$. By construction of h_φ, the only case when two transactions from different clauses write the same variable is when $\mathtt{var}(t) = \mathtt{var}(t')$. In particular, as t and t' belong to different clauses, there must exist indices i, i' s.t. $x = \mathtt{var}(t)^{\mathtt{sign}(t)}_{(i',i)}$ and $\mathsf{tr}(r_x) = S^j_i$. Hence, there is only one candidate for transaction t': t^j_i if $\mathtt{sign}(t^j_i) = \mathtt{sign}(t') = \mathtt{opsign}(t)$

and $\neg t_i^j$ otherwise. Therefore, as $t', \mathsf{tr}(r_x)$ belong to the same clause and t' is a $(\mathsf{so} \cup \overline{\mathsf{wr}})$-predecessor of $\mathsf{tr}(r_x)$, we conclude that $(t', S_i^j) \in \overline{\mathsf{wr}}_{\mathtt{var}(t')_i}$.

To reach a contradiction, we find a pair of distinct transactions $\tilde{t}, \hat{t}$ in the $\mathsf{pco}_{h_\varphi^\alpha}$-cycle from different clauses but associated to the same variable. First, as (t', t) is part of the $\mathsf{pco}_{h_\varphi^\alpha}$-cycle, there exists a $\mathsf{pco}_{h_\varphi^\alpha}$-predecessor of t', t'' s.t. $(t'', t') \in \mathsf{pco}_{h_\varphi^\alpha}$ is part of the $\mathsf{pco}_{h_\varphi^\alpha}$-cycle. As we mentioned before, $(t'', t') \in \mathsf{pco}_{h_\varphi^\alpha} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$. Then, there must exist a key y and a read event r_y s.t. t'' writes y, $(t', r_y) \in \overline{\mathsf{wr}}_y$ and $(t'', r) \in (\mathsf{so} \cup \overline{\mathsf{wr}}); \mathsf{po}^*$. Two cases arise:

- t'' is not associated to clause i: In this case, as both t', t'' write variable y, by construction of h_φ we observe that $\mathtt{var}(t'') = \mathtt{var}(t')$. Thus, we denote $\tilde{t} = t''$ and $\hat{t} = t'$.
- t'' is associated to clause i: In this case, $t'' \neq \neg t'$ as no transaction in h have both t' and $\neg t'$ as $(\mathsf{so} \cup \overline{\mathsf{wr}})$-predecessors. Hence, as no clause has two literals referring to the same variable, $\mathtt{var}(t') \neq \mathtt{var}(t'')$. Thus, as t'' and t' have one common key, we deduce that $t'' = \neg t_i^{j-1 \bmod 3}$ and $y = c_i^{j-1 \bmod 3}$. Thus, as $(t', r) \in \overline{\mathsf{wr}}_{c_i^{j-1 \bmod 3}}$, we can conclude that $\mathsf{tr}(r_y) = S_i^{j-1 \bmod 3}$ and $(t'', S_i^{j-1 \bmod 3}) \in \overline{\mathsf{wr}}_{\mathtt{var}(t'')_i}$. As $t'' \neq t$, there must exist a transaction t''' s.t. $(t''', t'') \in \mathsf{pco}_{h_\varphi^\alpha}$ belongs to the $\mathsf{pco}_{h_\varphi^\alpha}$-cycle. Again, we observe two cases:
    - t''' is not associated to clause i: In this case, by an analogous argument, we observe that $\mathtt{var}(t''') = \mathtt{var}(t'')$. Thus, we denote $\tilde{t} = t'''$ and $\hat{t} = t''$.
    - t''' is associated to clause i: By the same reasoning as before, $t''' = \neg t_i^{j-2 \bmod 3}$ and $(t''', S_i^{j-2 \bmod 3}) \in \overline{\mathsf{wr}}_{\mathtt{var}(t''')_i}$. Moreover, as $t''' \neq t$, there must exist a transaction t'''' s.t. $(t'''', t''') \in \mathsf{pco}_{h_\varphi^\alpha}$ belongs to the $\mathsf{pco}_{h_\varphi^\alpha}$-cycle. Moreover, t'''' is not associated to clause i, as, once more, we would deduce that $t'''' = \neg t_i^{j-3 \bmod 3}$ and that $(t'''', S_i^{j-3 \bmod 3}) \in \overline{\mathsf{wr}}_{\mathtt{var}(t'''')_i}$; which is impossible as by the construction of h_φ^α is satisfied. Hence, t'''' and t''' belong to different clauses and $\mathtt{var}(t'''') = \mathtt{var}(t''')$. We denote in this case $\tilde{t} = t''''$ and $\hat{t} = t'''$.

Finally, we reach a contradiction with the help of Lemmas 23 and 22. On one hand, by the choice of transactions $\hat{t}$ and $\tilde{t}$, we know that $\mathtt{var}(\hat{t}) = \mathtt{var}(\tilde{t})$ and there exist indices $\tilde{i}, \tilde{j}$ s.t. $\tilde{t}$ deletes $\mathtt{var}(l_{\tilde{i}}^{\tilde{j}})$. Moreover, $\hat{t} \neq \neg \tilde{t}_{\tilde{i}}^{\tilde{j}+1 \bmod 3}$ as they belong to different clauses. Thus, as $(\tilde{t}, \hat{t}) \in \mathsf{pco}_{h_\varphi^\alpha} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$, by Lemma 22 we deduce that $(\tilde{t}, S_{\tilde{i}}^{\tilde{j}}) \in \overline{\mathsf{wr}}_{\mathtt{var}(\tilde{t})_i}$. On the other hand, we also know that there exist indices $\hat{i}, \hat{j}$ s.t. $\hat{t}$ is associated to the literal $l_{\hat{i}}^{\hat{j}}$ and $(\hat{t}, S_{\hat{i}}^{\hat{j}}) \in \overline{\mathsf{wr}}_{\mathtt{var}(t)_{\hat{i}}}$. Hence, by construction of h_φ^α, as $\mathtt{var}(\hat{t}) = \mathtt{var}(\tilde{t})$, $(\tilde{t}, S_{\tilde{i}}^{\tilde{j}}) \in \overline{\mathsf{wr}}_{\mathtt{var}(\tilde{t})_i}$ and $(\hat{t}, {}_{\hat{i}}^{\hat{j}}) \in \overline{\mathsf{wr}}_{\mathtt{var}(\hat{t})_{\hat{i}}}$, we deduce that $\mathtt{sign}(\hat{t}) = \mathtt{sign}(\tilde{t})$. However, by Lemma 23, we deduce that $(\tilde{t}, \hat{t}) \notin \mathsf{pco}_{h_\varphi^\alpha} \setminus (\mathsf{so} \cup \overline{\mathsf{wr}})^+$. This contradicts that $(\tilde{t}, \hat{t}) h_\varphi^\alpha$ is part of the $\mathsf{pco}_{h_\varphi^\alpha}$-cycle. Thus, the initial hypothesis, that $\mathsf{pco}_{h_\varphi^\alpha}$ is cyclic, is false. In conclusion, $\mathsf{pco}_{h_\varphi^\alpha}$ is acyclic, so h_φ is consistent as h_φ^α is a consistent witness of h_φ.

□

Lemma 25. *For every boolean formula φ, if there is a consistent witness of h, then φ is satisfiable.*

Proof. Let $\overline{h} = (T, \mathsf{so}, \overline{\mathsf{wr}})$ be a consistent witness of h_φ. Hence, by Theorem 2, the relation $\mathsf{pco}_{\overline{h}} = \textsc{saturate}(\overline{h}, (\mathsf{so} \cup \overline{\mathsf{wr}})^+)$ is acyclic. We use this fact to construct a satisfying assignment of φ. Let us call u_i^j to the transaction s.t. $(u_i^j, S_i^j) \in \overline{\mathsf{wr}}_{\mathtt{var}(l_i^j)_i}$. Note that by construction of h_φ, u_i^j $\mathsf{deletes}$ $\mathtt{var}(l_i^j)_i$, so u_i^j is either t_i^j or $\neg t_i^j$.

We first prove that for every pair of pairs of indices i, i', j, j', if $\mathtt{var}(u_i^j) = \mathtt{var}(u_{i'}^{j'})$ then $\mathtt{sign}(u_i^j) = \mathtt{sign}(u_{i'}^{j'})$. By contradiction, let $u_i^j, u_{i'}^{j'}$ be a pair of transactions s.t. $\mathtt{var}(u_i^j) = \mathtt{var}(u_{i'}^{j'})$ and $\mathtt{sign}(u_i^j) \neq \mathtt{sign}(u_{i'}^{j'})$. In such case, $\mathtt{opsign}(u_i^j) = \mathtt{sign}(u_{i'}^{j'})$. Thus, both transactions write $\mathtt{var}(u_i^j)_{(i',i)}^{\mathtt{opsign}(u_i^j)}$ and $\mathtt{var}(u_i^j)_{(i,i')}^{\mathtt{sign}(u_i^j)}$. By axiom $\mathsf{Read\ Committed}$, as $(u_{i'}^{j'}, S_i^j) \in \overline{\mathsf{wr}}_{\mathtt{var}(u_i^j)_{(i',i)}^{\mathtt{opsign}(u_i^j)}}$ and $(u_i^j, S_i^j) \in \overline{\mathsf{wr}}$, we conclude that $(u_i^j, u_{i'}^{j'}) \in \mathsf{pco}_{\overline{h}}$. By a symmetric argument using $\mathtt{var}(u_i^j)_{(i,i')}^{\mathtt{sign}(u_i^j)}$ we deduce that $(u_{i'}^{j'}, u_i^j) \in \mathsf{pco}_{\overline{h}}$. However, this is impossible as $\mathsf{pco}_{\overline{h}}$ is acylclic; so we conclude that indeed $\mathtt{sign}(u_i^j) = \mathtt{sign}(u_{i'}^{j'})$.

Next, we construct a map that assign at each variable in φ a value 0 or 1. Let $\alpha_h : \mathsf{Vars}(\varphi) \to \{0, 1\}$ be the map that assigns for each variable $\mathtt{var}(l_i^j)$ the value 1 if $\mathtt{sign}(u_i^j) = +$ and 0 if $\mathtt{sign}(u_i^j) = -$. Note that this map is well defined as, by the previous paragraph, if two literals $l_i^j, l_{i'}^{j'}$ share variable, then their respective transactions $u_i^j, u_{i'}^{j'}$ have the same sign.

Finally, we prove that φ is satisfied with this assignment. By construction of α_h, for every pair of indices i, j, $l_i^j[\alpha_h(\mathtt{var}(l_i^j))/\mathtt{var}(l_i^j)]$ is true if and only if $(t_i^j, S_i^j) \in \overline{\mathsf{wr}}_{\mathtt{var}(l_i^j)}$. Moreover, as $\mathsf{pco}_{\overline{h}}$ is acyclic, by Lemma 21, we know that for each i there exists a j s.t. $u_i^j \neq \neg t_i^j$. Hence, for this j, u_i^j must be t_i^j as u_i^j is either t_i^j or $\neg t_i^j$. Therefore, every clause is satisfied using α_h as assignment; so φ is satisfiable.

□

B.4 Proof of Theorem 5.

Theorem 5. *Let h be a client history whose isolation configuration is defined using* $\{\mathtt{SER}, \mathtt{SI}, \mathtt{PC}, \mathtt{RA}, \mathtt{RC}\}$. *Algorithm 3 returns* true *if and only if h is consistent.*

The proof of Theorem 5 is a consequence of Lemmas 27 and 30.

Lemma 26. *Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a client history, $P = (T_P, M_P)$ be a consistent prefix of h and $t \in T \setminus T_P$. If $(P \cup \{t\}) \in \mathtt{seen}$ then* EXPLORECONSISTENTPREFIXES$(h, P \cup \{t\})$ *returns* false.

Proof. If $(P \cup \{t\}) \in \mathtt{seen}$, then $P \cup \{t\}$ has been to $\mathtt{seen}$ added at line 6 of Algorithm 4. To execute such instruction, the condition at line 4, EXPLORECONSISTENTPREFIXES$(h, P \cup \{t\})$ returns true, does not hold; which let us conclude the result. □

Lemma 27. *Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a client history whose isolation configuration is stronger than* $\mathtt{RC}$. *If h is consistent, Algorithm 3 returns* true.

Proof. Let h be a consistent history that satisfies the hypothesis of the Lemma. As h is consistent, let $\overline{h} = (T, \mathsf{so}, \overline{\mathsf{wr}})$ be a witness of h and let $\xi = (\overline{h}, \mathsf{co})$ be a consistent execution of $\overline{h}$. We first reduce the problem to prove that Algorithm 4 returns true on a particular witness of h, a history $\hat{h}$ s.t. $h \subseteq \hat{h} \subseteq \overline{h}$.

First, let pco, E_h and X_h be defined as in Algorithm 3 at lines 2-4. As $\overline{h}$ is consistent, for every $\mathtt{read}$ event r and a variable x s.t. $\mathsf{wr}_x^{-1}(r)\uparrow$, $\overline{\mathsf{wr}}_x^{-1}(r)\downarrow$ and $\mathtt{WHERE}(r)(\mathtt{value}_{\mathsf{wr}}(t_x^r, x)) = 0$; where $t_x^r = \overline{\mathsf{wr}}_x^{-1}(r)$.

On one hand, if E_h is empty, X_h is empty as well. In such case, we denote $\hat{h} = h$. On the other hand, if $E_h \neq \emptyset$, for every $(r, x) \in E_h$ we know that the transaction t_x^r belongs to $\mathsf{0}_x^r$. Therefore, $X_h \neq \emptyset$. Thus, let f be the map that assigns for every pair $(r, x) \in E_h$ the transaction t_x^r; and let $\hat{h} = (T, \mathsf{so}, \hat{\mathsf{wr}})$ be the history s.t. $\hat{h} = h \bigoplus_{(r,x) \in E_h} \mathsf{wr}_x(f(r, x), r)$. We observe that the fact that $\hat{h}$ is a history and $\overline{h}$ witnesses $\hat{h}$'s consistency using co is immediate as $\mathsf{wr} \subseteq \hat{\mathsf{wr}} \subseteq \overline{\mathsf{wr}}$. Note that in both cases, the condition at line 6 does not hold. Therefore, to prove that Algorithm 3 returns true it suffices to prove that EXPLORECONSISTENTPREFIXES$(\hat{h}, \emptyset)$ returns true.

We define an inductive sequence of prefixes based on co and show that they represent recursive calls to Algorithm 4. As a base case, let P_0 be the prefix with only $\mathtt{init}$ as transaction. Assuming that for every $j, 0 \leq j \leq i$, P_i is defined, let $P_{i+1} = P_i \cup \{t_i\}$; where t_i is the i-th transaction of T according to co. By construction of co, $\mathsf{pco} \subseteq \mathsf{co}$. Hence, Property 1 immediately holds. Moreover, as co witnesses $\overline{h}$'s consistency, Property 2 also holds; so $P_i \triangleright_{t_{i+1}} P_{i+1}$.

We conclude showing by induction on the number of transactions that are not in the prefix that for every $i, 0 \leq i \leq |T|$, EXPLORECONSISTENTPREFIXES$(\hat{h}, P_i)$ returns true.

- Base case: The base case is $i = |T|$. In such case, $P_{|T|}$ contains all transactions in T. Therefore, the condition at line 2 in Algorithm 4 holds and the algorithm returns true.

- Inductive case: The inductive hypothesis guarantees that for every $k, i \leq k \leq |T|$, $\textsc{exploreConsistentPrefixes}(\hat{h}, P_i)$ returns $\mathtt{true}$ and we show that $\textsc{exploreConsistentPrefixes}(\hat{h}, P_{i-1})$ also returns $\mathtt{true}$. By definition of P_i, $T_{P_i} = T_{P_{i-1}} \cup \{t_i\}$. In particular, $|P_i| \neq |T|$ and $P_{i-1} \triangleright_{t_{i+1}} P_i$. In addition, by induction hypothesis, we know that $\textsc{exploreConsistentPrefixes}(\hat{h}, P_i)$ returns $\mathtt{true}$. Hence, by Lemma 26, $P_i \notin \mathtt{seen}$. Altogether, we deduce that $\textsc{exploreConsistentPrefixes}(\hat{h}, P_{i-1})$ returns $\mathtt{true}$.

□

Lemma 28. *Let $\hat{h} = (T, \mathsf{so}, \widehat{\mathsf{wr}})$ be a client history and $P = (T_P, M_P)$ be a consistent prefix. If $\textsc{exploreConsistentPrefixes}(\hat{h}, P)$ returns $\mathtt{true}$, there exist distinct transactions $t_i \in T\ T_P$ and a collection of consistent prefixes $P_i = (T_P, M_P)$ s.t. $P_i = P_{i-1} \cup \{t_i\}$, $P_{i-1} \triangleright_{t_i} P_i$ and $\textsc{exploreConsistentPrefixes}(\hat{h}, P_i)$ returns $\mathtt{true}$; where $|T_P| < i \leq |T|$ and $P_{|T_P|} = P$.*

Proof. Let $\hat{h}$ be a client history and $P = (T_P, M_P)$ be a consistent prefix s.t. $\textsc{exploreConsistentPrefixes}(\hat{h}, P)$ returns $\mathtt{true}$. We prove the result by induction on the number of transactions not present in T_P. The base case, when $|T_P| = |T|$, immediately holds as $T \setminus T_P = \emptyset$. Let us assume that the inductive hypothesis holds for any prefix containing k transactions and let us show that it also holds for every consistent prefix with $k-1$ transactions. Let us thus assume that $|T_P| = k-1$. As $\textsc{exploreConsistentPrefixes}(\hat{h}, P)$ returns $\mathtt{true}$, it must reach line 5 in Algorithm 4. Hence, there must exist a transaction $t_k \in T \setminus T_P$ s.t. $P \triangleright_{t_k} (P \cup \{t_k\})$ and $\textsc{exploreConsistentPrefixes}(\hat{h}, P \cup \{t_k\})$ returns $\mathtt{true}$. By induction hypothesis on $P \cup \{t_k\} = (T_k, M_k)$, there exist a distinct transactions $t_i \in T \setminus T_k$ and a collection consistent prefixes P_i s.t. $P_i = P_{i-1} \cup \{t_i\}$, $P_{i-1} \triangleright_{t_i} P_i$ and $\textsc{exploreConsistentPrefixes}(\hat{h}, P_i)$ returns $\mathtt{true}$; where $k < i \leq |T|$ and $P_k = P \cup \{t_k\}$. Thus, the inductive step holds thanks to prefix P_k. □

Lemma 29. *Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a client history and let pco be the relation defined as at line 2 in Algorithm 4. If $\textsc{checkConsistency}(h)$ returns $\mathtt{true}$, there exists an extension $\hat{h} = (T, \mathsf{so}, \widehat{\mathsf{wr}})$ of h s.t. for every $\mathtt{read}$ event r, variable x and transaction t, (1) if $(t, r) \in \widehat{\mathsf{wr}}_x \setminus \mathsf{wr}_x$ then $t \in \mathsf{0}_x^r(\mathsf{pco})$, (2) if $\widehat{\mathsf{wr}}_x^{-1}(r) \uparrow$ then $\mathsf{1}_x^r(\mathsf{pco}) = \emptyset$, and (3) $\textsc{exploreConsistentPrefixes}(\hat{h}, \emptyset)$ returns $\mathtt{true}$.*

Proof. Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a client history s.t. $\textsc{checkConsistency}(h)$ returns $\mathtt{true}$ and let pco, E_h, X_h be the objects described in lines 2-4 in Algorithm 3. If there exists a pair $(r, x) \in E_h$ for which $\mathsf{0}_x^r(\mathsf{pco}) = \emptyset$, $\textsc{checkConsistency}(h)$ returns false. Hence, E_h is empty if and only if X_h is empty. If $E_h = \emptyset$, Algorithm 3 executes line 7. Thus, taking $\hat{h} = h$, conditions (1), (2) and (3) trivially hold. Otherwise, Algorithm 3 executes line 8. Once again, as $\textsc{exploreConsistentPrefixes}(h, \emptyset)$ returns $\mathtt{true}$, there must exists $f \in X_h$ s.t. $\textsc{exploreConsistentPrefixes}(\hat{h}, \emptyset)$ returns $\mathtt{true}$; where $\hat{h} = \bigoplus_{(r,x) \in E_h} \mathsf{wr}_x(f(r, x), r)$. Thanks to the definition of f and $\hat{h}$ conditions (1), (2) and (3) are satisfied. □

Lemma 30. *Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a client history whose isolation configuration is composed of* $\{\mathtt{SER}, \mathtt{SI}, \mathtt{PC}, \mathtt{RC}\}$ *isolation levels. If Algorithm 4 returns* $\mathtt{true}$*, h is consistent.*

Proof. Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a client history s.t. $\textsc{checkConsistency}(h)$ returns $\mathtt{true}$ and let pco, E_h and X_h be defined as at lines 2-4 in Algorithm 3. By Lemma 29, there exists an extension of h $\hat{h} = (T, \mathsf{so}, \widehat{\mathsf{wr}})$ s.t. for every $\mathtt{read}$ event r, variable x and transaction t, (1) if $\widehat{\mathsf{wr}}_x^{-1}(r) \uparrow$ then $\mathsf{1}_x^r(\mathsf{pco}) = \emptyset$, (2) if $(t, r) \in \widehat{\mathsf{wr}}_x \setminus \mathsf{wr}_x$ then $t \in \mathsf{0}_x^r(\mathsf{pco})$ and (3) $\textsc{exploreConsistentPrefixes}(\hat{h}, \emptyset)$ returns $\mathtt{true}$. By Lemma 28 applied on $\hat{h}$ and $\emptyset$, there exist distinct transactions $t_i \in T$ and a collection of prefixes of h, $P_i = (T_i, M_i)$, s.t. $P_i = P_{i-1} \cup \{t_i\}$, $P_{i-1} \triangleright_{t_i} P_i$ and $\textsc{exploreConsistentPrefixes}(\hat{h}, P_i)$ returns $\mathtt{true}$; where $P_0 = \emptyset$ and $0 < i \leq |T|$. Let co be the total order based on the aforementioned transactions t_i, i.e. $\mathsf{co} = \{(t_i, t_j) \mid i < j\}$. We construct a full history that extends $\hat{h}$ employing co and taking into account the isolation level of each transaction.

For every $\mathtt{read}$ event r, key x and visibility relation $\mathsf{v} \in \mathsf{vis}(\mathsf{iso}(h)(\mathsf{tr}(r)))$, let t_{v}, t_x^r be the transactions defined as follows:

$$t_{\mathsf{v}}^x = \max_{\mathsf{co}}\{t' \in T \mid t' \text{ writes } x \ \wedge\ \mathsf{v}(\mathsf{co})(t', r, x)\}$$
$$t_x^r = \max_{\mathsf{co}}\{t_{\mathsf{v}}^x \mid \mathsf{v} \in \mathsf{vis}(\mathsf{iso}(h)(\mathsf{tr}(r)))\} \tag{10}$$

Note that if v is a visibility relation associated to an axiom from $\mathtt{SER}, \mathtt{SI}, \mathtt{PC}, \mathtt{RA}$ and $\mathtt{RC}$ isolation levels, transactions t_{v}^x and t_x^r are well-defined as $\mathsf{v}(\mathtt{init}, r, x)$ holds. Thus, let $\overline{\mathsf{wr}}_x = \widehat{\mathsf{wr}}_x \cup \{(t_x^r, r) \mid \widehat{\mathsf{wr}}_x^{-1}(r) \uparrow\}$ and $\overline{\mathsf{wr}} = \bigcup_{x \in \mathsf{Keys}} \overline{\mathsf{wr}}_x$. As $\overline{\mathsf{wr}}_x^{-1}$ is a total function and $\overline{\mathsf{wr}}_x^{-1}(r)$ writes x we can conclude that $\overline{h} = (T, \mathsf{so}, \overline{\mathsf{wr}})$ is a full history.

We prove that $\overline{h}$ is also a witness of h. For that, we show that for every read event r, every key x and every transaction t, if $(t, r) \in \overline{\mathsf{wr}}_x \setminus \mathsf{wr}_x$, $t \in \mathsf{0}_x^r(\mathsf{pco})$. Two cases arise: $(t, r) \in \widehat{\mathsf{wr}}_x \setminus \mathsf{wr}_x$ and $(t, r) \in \overline{\mathsf{wr}}_x \setminus \widehat{\mathsf{wr}}_x$. The first case is quite straightforward, as if $(t, r) \in \widehat{\mathsf{wr}}_x \setminus \mathsf{wr}_x$, by Property (1) of Lemma 29, $t \in \mathsf{0}_x^r(\mathsf{pco})$. The second case, $(t, r) \in \overline{\mathsf{wr}}_x \setminus \widehat{\mathsf{wr}}_x$, is slightly more subtle. First, for every isolation level considered, if $(t, r) \in \overline{\mathsf{wr}}_x$ then $(t, \mathsf{tr}(r)) \in \mathsf{pco}$. Next, as $\textsc{checkConsistency}(h)$ returns $\mathtt{true}$, the condition at line 5 does not hold. Hence, as pco is acyclic, we deduce that $(\mathsf{tr}(r), t) \notin \mathsf{pco}$. In addition, as $(t, r) \in \overline{\mathsf{wr}}_x \setminus \widehat{\mathsf{wr}}_x$, $\widehat{\mathsf{wr}}_x^{-1}(r) \uparrow$. By Property (2) of Lemma 29 employed during $\hat{h}$'s construction, we deduce that $\mathsf{1}_x^r(\mathsf{pco}) = \emptyset$. In conclusion, as $(\mathsf{tr}(r), t) \notin \mathsf{pco}$ and $\mathsf{1}_x^r(\mathsf{pco}) = \emptyset$, we conclude that $t \in \mathsf{0}_x^r(\mathsf{pco})$.

Finally, we prove that co witnesses that $\overline{h}$ is consistent. Let r be a $\mathtt{read}$ event, x be a key and t_1, t_2 be transactions s.t. $(t_1, r) \in \overline{\mathsf{wr}}_x$ and t_2 writes x. We prove that if there exists $\mathsf{v} \in \mathsf{vis}(\mathsf{iso}(\overline{h})(\mathsf{tr}(r)))$ s.t. $\mathsf{v}(\mathsf{co})(t_2, r, x)$ holds in $\overline{h}$ then $(t_2, t_1) \in \mathsf{co}$; which by Definition 6, we know it implies that $\overline{h}$ is consistent. Note that if $(t_1, r) \in \overline{\mathsf{wr}} \setminus \widehat{\mathsf{wr}}$, by definition of t_x^r the statement immediately holds; so we can assume without loss of generality that $(t_1, r) \in \widehat{\mathsf{wr}}_x$.

First, we note that proving that whenever $\mathsf{v}(\mathsf{co})(t_2, r, x)$ holds in $\overline{h}$, then $(t_2, t_1) \in \mathsf{co}$ is equivalent to prove that whenever $\mathsf{v}(\mathsf{co})(t_2, r, x)$ holds in $\overline{h}$, then $t_1 \notin T_{i-1}$; where i is the index of the transaction in T s.t. $t_2 = t_i$.

For every $i, 1 \leq i \leq |T|$ $P_{i-1} \triangleright_{t_i} P_i$, $\mathsf{so} \cup \mathsf{\hat{w}r} \subseteq \mathsf{co}$. Thus, by Definition 6, it suffices to show that for every read event r, $C_{\mathsf{iso}(h)(\mathsf{tr}(r))}(\mathsf{pco})(r)$ holds. For that, let $\mathsf{p\hat{c}o} = \mathsf{FIX}(\lambda R : \textsc{saturate}(\hat{h}, R))(\mathsf{so} \cup \mathsf{\hat{w}r})^+$ be the partial commit order implied by $\hat{h}$.

As $\mathsf{iso}(h)$ is composed of $\{\mathtt{SER}, \mathtt{SI}, \mathtt{PC}, \mathtt{RA}, \mathtt{RC}\}$ isolation levels and $P_{i-1} \triangleright_{t_2} P_i$, by Property 2 of Definition 10, it suffices to prove that whenever $\mathsf{v}(\mathsf{co})(t_2, r, x)$, if $v \neq \mathsf{Conflict}$ then $v(\mathsf{p\hat{c}o}_{t_2}^{P_i})(t, r, x)$ holds in $\hat{h}$, while if $v = \mathsf{Conflict}$, that there exists $t' \in T_{i-1}$ s.t. $v(\mathsf{p\hat{c}o}_{t_2}^{P_i})(t', r, x)$ holds in $\hat{h}$; where $\mathsf{p\hat{c}o}_{t_2}^{P_i}$ is obtained by applying Table 1 on $\mathsf{p\hat{c}o}$. We analyze five different cases:

- $\mathsf{iso}(\overline{h})(\mathsf{tr}(r)) = \mathtt{SER}$: In this case, $\mathsf{Serializability}(\mathsf{co})(t_2, r, x)$ holds in $\overline{h}$ if and only if $(t_2, \mathsf{tr}(r)) \in \mathsf{co}$. As $\mathsf{p\hat{c}o}_{t_2}^{P_i}$ totally orders t_2 and every other transaction in T and $\mathsf{p\hat{c}o}_{t_2}^{P_i} \subseteq \mathsf{co}$, we deduce that $(t_2, \mathsf{tr}(r)) \in \mathsf{p\hat{c}o}_{t_2}^{P_i}$. Hence, $\mathsf{Serializability}(\mathsf{p\hat{c}o}_{t_2}^{P_i})(t_2, r, x)$ holds in $\hat{h}$.
- $\mathsf{iso}(\overline{h})(\mathsf{tr}(r)) = \mathtt{SI}$: Two disjoint sub-cases arise:
  - $\mathsf{Conflict}(\mathsf{co})(t_2, r, x)$ holds in $\overline{h}$: This happens if and only if there exists a transaction t_3 and a key $y \in \mathsf{Keys}$ s.t. t_3 writes y, $\mathsf{tr}(r)$ writes y, $(t_2, t_3) \in \mathsf{co}^*$ and $(t_3, \mathsf{tr}(r)) \in \mathsf{co}$. Let j be the index s.t. $t_3 = t_j$. Then, as $\mathsf{p\hat{c}o}_{t_3}^{P_j}$ totally orders t_3 and every other transaction and $\mathsf{p\hat{c}o}_{t_3}^{P_j} \subseteq \mathsf{co}$, $(t_2, t_3) \in (\mathsf{p\hat{c}o}_{t_3}^{P_j})^*$ and $(t_3, \mathsf{tr}(r)) \in \mathsf{p\hat{c}o}_{t_3}^{P_j}$. Thus, $\mathsf{Conflict}(\mathsf{p\hat{c}o}_{t_3}^{P_j})(t_2, r, x)$ holds in $\hat{h}$.
  - $\mathsf{Prefix}(\mathsf{co})(t_2, r, x)$ holds in $\overline{h}$ but $\mathsf{Conflict}(\mathsf{co})(t_2, r, x)$ does not: We observe that $\mathsf{Prefix}(\mathsf{co})(t_2, r, x)$ holds in $\overline{h}$ if there exists a transaction t_3 s.t. $(t_2, t_3) \in \mathsf{co}^*$ and $(t_3, \mathsf{tr}(r)) \in \mathsf{so} \cup \overline{\mathsf{wr}}$. If $(t_3, \mathsf{tr}(r)) \in \overline{\mathsf{wr}} \setminus (\mathsf{so} \cup \mathsf{\hat{w}r})$, by Equation (10) there exist $y \in \mathsf{Keys}$ and $v \in \mathsf{vis}(\mathtt{SI})$ s.t. $v(\mathsf{co})(t_3, r, y)$ holds in $\hat{h}$. Note that $v \neq \mathsf{Conflict}$ as otherwise $\mathsf{Conflict}(\mathsf{co})(t_2, r, x)$ would hold in $\overline{h}$. Hence, $v = \mathsf{Prefix}$ and by transitivity of co, we conclude that $\mathsf{Prefix}(\mathsf{co})(t_2, r, x)$ holds in $\hat{h}$. As $\mathsf{p\hat{c}o}_{t_2}^{P_i}$ totally orders t_2 with respect every other transaction in t_2 and $\mathsf{p\hat{c}o}_{t_2}^{P_i} \subseteq \mathsf{co}$, we conclude that $\mathsf{Prefix}(\mathsf{p\hat{c}o}_{t_2}^{P_i})(t_2, r, x)$ holds in $\hat{h}$.
- $\mathsf{iso}(\overline{h})(\mathsf{tr}(r)) = \mathtt{PC}$: In this case, $\mathsf{Prefix}(\mathsf{co})(t_2, r, x)$ holds in $\overline{h}$ if and only if there exists a transaction t_3 s.t. $(t_2, t_3) \in \mathsf{co}^*$ and $(t_3, \mathsf{tr}(r)) \in \mathsf{so} \cup \overline{\mathsf{wr}}$. If $(t_3, \mathsf{tr}(r)) \in \overline{\mathsf{wr}} \setminus (\mathsf{so} \cup \mathsf{\hat{w}r})$, by Equation (10) there exist $y \in \mathsf{Keys}$ and $v \in \mathsf{vis}(\mathtt{SI})$ s.t. $v(\mathsf{co})(t_3, r, y)$ holds in $\hat{h}$. Hence, by transitivity of co, we conclude that $\mathsf{Prefix}(\mathsf{co})(t_2, r, x)$ holds in $\hat{h}$. As $\mathsf{p\hat{c}o}_{t_2}^{P_i}$ totally orders t_2 with respect every other transaction in t_2 and $\mathsf{p\hat{c}o}_{t_2}^{P_i} \subseteq \mathsf{co}$, we conclude that $\mathsf{Prefix}(\mathsf{p\hat{c}o}_{t_2}^{P_i})(t_2, r, x)$ holds in $\hat{h}$.
- $\mathsf{iso}(\overline{h})(\mathsf{tr}(r)) = \mathtt{RA}$: In this case, $\mathsf{Read\ Atomic}(\mathsf{co})(t_2, r, x)$ holds in $\overline{h}$ if and only if $(t_2, \mathsf{tr}(r)) \in \mathsf{so} \cup \overline{\mathsf{wr}}$. We observe that by Equation (10), if $(t_2, \mathsf{tr}(r)) \in \overline{\mathsf{wr}} \setminus (\mathsf{so} \cup \mathsf{\hat{w}r})$, then $t_2 = t_x^r$ and $\mathsf{Read\ Atomic}(\mathsf{p\hat{c}o}_{t_2}^{P_i})(t_2, r, x)$ holds in $\hat{h}$. Hence, $(t_2, \mathsf{tr}(r)) \in \mathsf{so} \cup \mathsf{\hat{w}r}$; which is a contradiction. Thus, as $(t_2, \mathsf{tr}(r)) \in \mathsf{so} \cup \mathsf{\hat{w}r}$, $\mathsf{Read\ Atomic}(\mathsf{p\hat{c}o}_{t_2}^{P_i})(t_2, r, x)$ holds in $\hat{h}$.

- $\underline{\mathsf{iso}(\overline{h})(\mathsf{tr}(r)) = \mathtt{RC}}$: Similarly to the previous case, we observe that the formula $\mathsf{Read\ Committed}(\mathsf{co})(t_2, r, x)$ holds in $\overline{h}$ iff $(t_2, \mathsf{tr}(r)) \in (\mathsf{so} \cup \overline{\mathsf{wr}}); \mathsf{po}^*$. We observe that by Equation (10), if $(t_2, \mathsf{tr}(r)) \in \overline{\mathsf{wr}} \setminus (\mathsf{so} \cup \hat{\mathsf{wr}}); \mathsf{po}^*$, then $t_2 = t^r_x$ and $\mathsf{Read\ Committed}(\hat{\mathsf{pco}}^{P_i}_{t_2})(t_2, r, x)$ holds in $\hat{h}$. Therefore, $(t_2, r) \in \mathsf{so} \cup \hat{\mathsf{wr}}; \mathsf{po}^*$; which is a contradiction. Thus, as $(t_2, \mathsf{tr}(r)) \in \mathsf{so} \cup \hat{\mathsf{wr}}; \mathsf{po}^*$, $\mathsf{Read\ Committed}(\hat{\mathsf{pco}}^{P_i}_{t_2})(t_2, r, x)$ holds in $\hat{h}$.

□

Algorithm 5 Checking if $P \triangleright_t (P \cup \{t\})$ holds in h

```
1: function isConsistentExtension($h = (T, \mathsf{so}, \mathsf{wr})$, $P = (T_P, M_P)$, $t$)
                                                          ▷ We assume $t \notin T_P$.
2:     let $\mathsf{pco} = \mathsf{FIX}(\lambda R : \textsc{saturate}(h, R))(\mathsf{so} \cup \mathsf{wr})^+$
3:     if $\exists t' \in T \setminus T_P$ s.t. $(t', t) \in \mathsf{pco}$ then          ▷ Condition 1
4:         return false
5:     for all $r \in \mathsf{reads}(h)$ s.t. $\mathsf{tr}(r) \notin T_P \cup \{t\}, v \in \mathsf{vis}(\mathsf{iso}(h))(\mathsf{tr}(r))$ do
6:         if $\mathsf{vp}^P_v(t, r)$ does not hold in $h$ then                ▷ Condition 2
7:             return false
8:     return true
```

B.5 Proof of Theorem 6

Theorem 6. *For every client history h whose isolation configuration is composed of $\{\mathtt{SER}, \mathtt{SI}, \mathtt{PC}, \mathtt{RA}, \mathtt{RC}\}$ isolation levels, Algorithm 3 runs in $\mathcal{O}(|h|^{\#\mathtt{conf}(h)+\mathtt{width}(h)+9} \cdot \mathtt{width}(h)^{|\mathsf{Keys}|})$. Moreover, if no transaction employs* SI *isolation level, Algorithm 3 runs in $\mathcal{O}(|h|^{\#\mathtt{conf}(h)+\mathtt{width}(h)+8})$.*

The proof of Theorem 6 is split in two Lemmas: Lemma 34 analyzes the complexity of Algorithm 4 while Lemma 35 relies on the previous result to conclude the complexity of Algorithm 3.

Lemma 31. *Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a history, $P = (T_P, M_P)$ be a consistent prefix of h and $t \in T \setminus T_P$ be a transaction. Algorithm 5 returns* true *if and only if $P \triangleright_t (P \cup \{t\})$.*

Proof. Clearly, $P \cup \{t\}$ is an extension of P.$\textsc{isConsistentExtension}(h, P, t)$ returns true if and only if conditions at lines 3 and lines 5 in Algorithm 5 hold. This is equivalent to respectively satisfy Properties 1 and 2 of Definition 10. By Definition 10, this is equivalent to $P \triangleright_t (P \cup \{t\})$. □

Lemma 32. *Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a history and $k \in \mathbb{N}$ be a bound in $\mathsf{iso}(h)$. For any consistent prefix $P = (T_P, M_P)$ of h and any transaction $t \in T \setminus T_P$, Algorithm 5 runs in $\mathcal{O}(|h|^{k+3})$.*

Proof. We analyze the cost of Algorithm 5. First, as $\mathsf{pco} \subseteq T \times T$, by Lemma 7, line 2 runs in $\mathcal{O}(|h|^2 \cdot |h|^{k+1})$. Next, the condition at line 3 can be checked in $\mathcal{O}(|T|)$. Finally, the condition at line 5 can be checked in $\mathcal{O}(|T| \cdot k \cdot U)$; where U is an upper-bound on the complexity of checking $\mathsf{vp}^P_v(t, r)$. With the aid of Lemma 6, we deduce that $U \in \mathcal{O}(|h|^{k-2})$. Altogether, we conclude that Algorithm 5 runs in $\mathcal{O}(|h|^{k+3})$. □

Lemma 33. *Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a client history. If $\mathsf{iso}(h)$ is composed of $\{\mathtt{SER}, \mathtt{SI}, \mathtt{PC}, \mathtt{RA}, \mathtt{RC}\}$ isolation levels, then 5 is a bound of $\mathsf{iso}(h)$. Moreover, if no transaction has* SI *as isolation, 4 is a bound on $\mathsf{iso}(h)$.*

Proof. Let h be a history as described in the hypothesis. First, all isolation levels in the set $\{\mathtt{SER}, \mathtt{SI}, \mathtt{PC}, \mathtt{RA}, \mathtt{RC}\}$ employ at most two axioms. Moreover, every axiom described employs at most 5 quantifiers: three universal quantifiers and at most two existential quantifiers. Hence, 5 is a bound on $\mathsf{iso}(h)$. Note that $\mathsf{Conflict}$ is the only axiom employing two existential quantifiers; so if no transaction employs $\mathtt{SI}$, 4 bounds $\mathsf{iso}(h)$. □

Lemma 34. *Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a client history whose isolation configuration is composed of $\{\mathtt{SER}, \mathtt{SI}, \mathtt{PC}, \mathtt{RA}, \mathtt{RC}\}$ isolation levels. Algorithm 4 runs in $\mathcal{O}(|h|^{\mathtt{width}(h)+9} \cdot \mathtt{width}(h)^{|\mathsf{Keys}|})$. Moreover, if no transaction has $\mathtt{SI}$ as isolation level, Algorithm 4 runs in $\mathcal{O}(|h|^{\mathtt{width}(h)+8})$.*

Proof. For proving the result, we focus only on prefixes that are computable by Algorithm 4. Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a history. A prefix P of h is *computable* if either $P = \emptyset$ or there exist a transaction t and a prefix P' s.t. $P = P' \cup \{t\}$ and P' is computable.

Intuitively, computable prefixes represent recursive calls of Algorithm 4 when employed by Algorithm 3. Indeed, Algorithm 3 only employs Algorithm 4 at lines 7 and 11. In both cases, $P' = \emptyset$ is the initial call to Algorithm 4. Moreover, the condition at line 3 justifies the recursive definition.

On one hand, we observe that any call to Algorithm 4 is associated to a computable prefix and on the other hand, Algorithm 4 does not explore two equivalent computable prefix thanks to the global variable $\mathtt{seen}$ (line 4). Therefore, Algorithm 4 runs in $\mathcal{O}(N \cdot U)$; where N is the number of distinct equivalence class of prefixes of h and U is an upper-bound on the running time of Algorithm 4 on a fixed prefix without doing any recursive call.

We first compute an upper-bound of N. For any computable prefix P, we can deduce by induction on the length of P that there exists transactions $t_i \in T_P$ and a collection of computable prefixes of h, $P_i = (T_i, M_i)$ and transactions t_i s.t. $P_{|T_P|} = P$, $P_i = P_{i-1} \cup \{t_i\}$ and $P_{i-1} \triangleright_{t_i} P_i$; where $P_0 = \emptyset$ and $0 < i \leq |T_P|$. The base case is immediate as $|T_P| = 0$ implies that $T' = \emptyset$ while the inductive step can be simply obtained by applying the recursive definition of computable prefix.

Let $P = (T_P, M_P)$ be in what follows a computable prefix of h. We observe that both T_P and M_P are determined by its so-maximal transactions. Let $t, t' \in T$ be a pair of transactions s.t. $(t, t') \in \mathsf{so}$ and $t' \in T_P$. As $t' \in T_P$ there must exist an index $i, 1 \leq i \leq |T_P|$ s.t. $P_i = P_{i-1} \cup \{t'\}$. Therefore, as $P_{i-1} \triangleright_{t'} P_{i-1}$, $t \in T_{i-1} \subseteq T_P$. In particular, if t' is a so-maximal transaction in T_P, all its so-predecessors are also contained in T_P; and hence, T_P can be characterized by its so-maximal transactions. Moreover, by induction on the length of P we can prove that for every key x, $M_P(x)$ is a so-maximal transaction: the base case, $|T_P| = 0$ is immediate while the inductive step is obtained by the definition of $P \cup \{t''\}, t'' \notin T_P$. Hence, the number of computable prefixes of a history is in $\mathcal{O}(|T|^{\mathtt{width}(h)} \cdot \mathtt{width}(h)^{|\mathsf{Keys}|})$. Thus, $N \in \mathcal{O}(|h|^{\mathtt{width}(h)} \cdot \mathtt{width}(h)^{|\mathsf{Keys}|})$. Moreover, if no transaction employs $\mathtt{SI}$ as isolation level, prefixes with identical transaction set coincide. Hence, in such case, $N \in \mathcal{O}(|h|^{\mathtt{width}(h)})$.

We conclude the proof bounding U. If $|T_P| = |T|$, Algorithm 4 runs in $\mathcal{O}(1)$; so we can assume without loss of generality that $|T_P| \neq |T|$. In such case, U

represent the cost of executing lines 3-7 in Algorithm 4. Thus, $U \in \mathcal{O}((|T| - |T_P|) \cdot V)$; where V is the cost of checking $P \triangleright_t (P \cup \{t\})$ for a transaction $t \in T \setminus T_P$. By Lemma 31, Algorithm 5 can check if $P \triangleright_t (P \cup \{t\})$ and thanks to Lemma 32, Algorithm 5 runs in $\mathcal{O}(|h|^{k+3})$; where k is a bound on $\mathsf{iso}(h)$. Thus, $U \in \mathcal{O}(|h|^{k+4})$.

Thanks to Lemma 33, we conclude that Algorithm 4 runs in $\mathcal{O}(|h|^{\mathtt{width}(h)+9} \cdot \mathtt{width}(h)^{|\mathsf{Keys}|})$ and, if no transaction employs $\mathtt{SI}$ as isolation level, then it runs in $\mathcal{O}(|h|^{\mathtt{width}(h)+8})$. □

Lemma 35. *Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a client history whose isolation configuration is composed of $\{\mathtt{SER}, \mathtt{SI}, \mathtt{PC}, \mathtt{RA}, \mathtt{RC}\}$ isolation levels. Algorithm 3 runs in $\mathcal{O}(|h|^{\#\mathtt{conf}(h)+\mathtt{width}(h)+9} \cdot \mathtt{width}(h)^{|\mathsf{Keys}|})$. Moreover, if no transaction has $\mathtt{SI}$ as isolation level, Algorithm 3 runs in $\mathcal{O}(|h|^{\#\mathtt{conf}(h)+\mathtt{width}(h)+8})$.*

Proof. Let $h = (T, \mathsf{so}, \mathsf{wr})$ be a history satisfying the hypothesis of the Lemma. We decompose our analysis in two sections, the first one where we analyze the complexity of executing lines 2-4 and second one where we analyze the complexity of executing lines 5-12. We observe that by Lemma 33, 5 is a bound on $\mathsf{iso}(h)$.

In line 2, Algorithm 3 computes pco. On one hand, computing $(\mathsf{so} \cup \mathsf{wr})^+$ is in $\mathcal{O}(|T|^3)$. On the other hand, as $\mathsf{pco} \subseteq T \times T$ and by Lemma 7, executing $\textsc{saturate}(h, (\mathsf{so} \cup \mathsf{wr})^+)$ is in $\mathcal{O}(|h|^6)$; we deduce that computing pco after compute $(\mathsf{so} \cup \mathsf{wr})^+$ is in $\mathcal{O}(|h|^8)$.

In line 3, Algorithm 3 computes E_h. As wr is acyclic, for a given key x and transaction t, $\mathtt{value}_{\mathsf{wr}}(t, x) \in \mathcal{O}(|T|)$. Therefore, computing $1_x^r(\mathsf{pco})$ is in $\mathcal{O}(|T|)$ as we assume that for every $r \in \mathsf{Rows}$, computing $\mathtt{WHERE}(r)(v) \in \mathcal{O}(1)$. Thus, computing E_h is in $\mathcal{O}(|h|^3)$.

Finally, in line 4, Algorithm 3 computes X_h. Note that X_h can be seen is a $\bigtimes_{(r,x) \in E_h} 0_x^r(\mathsf{pco})$. Computing each $0_x^r(\mathsf{pco})$ set is in $\mathcal{O}(|T|)$; so computing all of them is in $\mathcal{O}(|T| \cdot |E_h|)$. As each set $0_x^r(\mathsf{pco})$ is a subset of T, applying the cartesian-product definition of X_h we can compute X_h in $\mathcal{O}(|T|^{|E_h|})$. Therefore, as $|E_h| = \#\mathtt{conf}(h)$, we conclude that computing X_h is in $\mathcal{O}(|h| \cdot \#\mathtt{conf}(h) + |h|^{\#\mathtt{conf}(h)})$ and that $|X_h| \in \mathcal{O}(|h|^{\#\mathtt{conf}(h)})$. Altogether, as $\#\mathtt{conf}(h) \leq |T|^2$, we deduce that computing lines 2-4 of Algorithm 3 is in $\mathcal{O}(|h|^8 + |h|^{\#\mathtt{conf}(h)})$.

Next, we analyze the complexity of executing lines 5-12. Four disjoint cases arise, one per boolean condition in Algorithm 3. The first one, checking if pco is cyclic (line 5), is in $\mathcal{O}(|h|)$. The second one, checking if $\exists (r, x) \in E_h$ s.t. $0_x^r(\mathsf{pco}) = \emptyset$ (line 6), clearly runs in $\mathcal{O}(\#\mathtt{conf}(h) \cdot |h|)$. The third one, checking if $E_h = \emptyset$ and executing Algorithm 4 is in $\mathcal{O}(\#\mathtt{conf}(h) + |h|^{\mathtt{width}(h)+9} \cdot \mathtt{width}(h)^{|\mathsf{Keys}|})$ thanks to Lemma 34.

Finally we analyze the last case, computing an extension of h for each mapping in X_h and then executing Algorithm 3 (lines 8-12). On one hand, computing each history is in $\mathcal{O}(|h|^3)$ as we require to define both $\mathsf{so} \subseteq T \times T$ and $\mathsf{wr} \subseteq \mathsf{Keys} \times T \times T$. On the other hand, as the size of each extension of h is in $\mathcal{O}(|h|)$, executing Algorithm 3 for a given history is in $\mathcal{O}(|X_h| \cdot |h|^{\mathtt{width}(h)+9} \cdot \mathtt{width}(h)^{|\mathsf{Keys}|})$ thanks again to Lemma 34. Altogether, for each mapping $f \in X_h$, executing lines 10-11 is in $\mathcal{O}(|h|^{\texttt{width}(h)+9} \cdot \texttt{width}(h)^{|\mathsf{Keys}|})$. As $|X_h| \in \mathcal{O}(|h|^{\#\texttt{conf}(h)})$, we conclude that executing this last case is in $\mathcal{O}(|h|^{\#\texttt{conf}(h)+\texttt{width}(h)+9} \cdot \texttt{width}(h)^{|\mathsf{Keys}|})$.

We then conclude that Algorithm 3 runs in $\mathcal{O}(|h|^{\#\texttt{conf}(h)+\texttt{width}(h)+9} \cdot \texttt{width}(h)^{|\mathsf{Keys}|})$. Moreover, if no transaction employs `SI` as isolation level, Lemma 34 allows us to deduce that in such case Algorithm 3 runs in $\mathcal{O}(|h|^{\#\texttt{conf}(h)+\texttt{width}(h)+9})$. □