On the Complexity of Checking Transactional Consistency

RANADEEP BISWAS, Universite de Paris, IRIF, CNRS, France
CONSTANTIN ENEA, Universite de Paris, IRIF, CNRS, France

Transactions simplify concurrent programming by enabling computations on shared data that are isolated from other concurrent computations and are resilient to failures. Modern databases provide different consistency models for transactions corresponding to different tradeoffs between consistency and availability. In this work, we investigate the problem of checking whether a given execution of a transactional database adheres to some consistency model. We show that consistency models like read committed, read atomic, and causal consistency are polynomial-time checkable while prefix consistency and snapshot isolation are NP-complete in general. These results complement a previous NP-completeness result concerning serializability. Moreover, in the context of NP-complete consistency models, we devise algorithms that are polynomial time assuming that certain parameters in the input executions, e.g., the number of sessions, are fixed. We evaluate the scalability of these algorithms in the context of several production databases.

CCS Concepts: • **General and reference → Validation**; • **Information systems → Key-value stores**; • **Theory of computation → Logic and verification**; • **Software and its engineering → Consistency**; **Dynamic analysis**; **Formal software verification**.

Additional Key Words and Phrases: transactional databases, consistency, axiomatic specifications, testing

ACM Reference Format:
Ranadeep Biswas and Constantin Enea. 2018. On the Complexity of Checking Transactional Consistency. 1, 1 (January 2018), 27 pages.

1 INTRODUCTION

Transactions simplify concurrent programming by enabling computations on shared data that are isolated from other concurrent computations and resilient to failures. Modern databases provide transactions in various forms corresponding to different tradeoffs between consistency and availability. The strongest level of consistency is achieved with *serializable* transactions [25] whose outcome in concurrent executions is the same as if the transactions were executed atomically in some order. Unfortunately, serializability carries a significant penalty on the availability of the system assuming, for instance, that the database is accessed over a network that can suffer from partitions or failures. For this reason, modern databases often provide weaker guarantees about transactions, formalized by weak consistency models, e.g., causal consistency [22] and snapshot isolation [11].

Implementations of large-scale databases providing transactions are difficult to build and test. For instance, distributed (replicated) databases must account for partial failures, where some components or the network can fail and produce incomplete results. Ensuring fault-tolerance relies on intricate protocols that are difficult to design and reason about. The black-box testing framework Jepsen [1] found a remarkably large number of subtle problems in many production distributed databases.

Testing a transactional database raises two issues: (1) deriving a suitable set of testing scenarios, e.g., faults to inject into the system and the set of transactions to be executed, and (2) deriving efficient algorithms for checking whether a given execution satisfies the considered consistency

Authors' addresses: Ranadeep Biswas, Universite de Paris, IRIF, CNRS, Paris, F-75013, France, ranadeep@irif.fr; Constantin Enea, Universite de Paris, IRIF, CNRS, Paris, F-75013, France, cenea@irif.fr.

2018. XXXX-XXXX/2018/1-ART $15.00
https://doi.org/

model. The Jepsen framework aims to address the first issue by using randomization, e.g., introducing faults at random and choosing the operations in a transaction randomly. The effectiveness of this approach has been proved formally in recent work [24]. The second issue is, however, largely unexplored. Jepsen checks consistency in a rather ad-hoc way, focusing on specific classes of violations to a given consistency model, e.g., dirty reads (reading values from aborted transactions). This problem is challenging because the consistency specifications are non-trivial and they cannot be checked using, for instance, standard local assertions added to the client's code.

Besides serializability, the complexity of checking correctness of an execution w.r.t. some consistency model is unknown. Checking serializability has been shown to be NP-complete [25], and checking causal consistency in a *non-transactional* context is known to be polynomial time [13]. In this work, we try to fill this gap by investigating the complexity of this problem w.r.t. several consistency models and, in case of NP-completeness, devising algorithms that are polynomial time assuming fixed bounds for certain parameters of the input executions, e.g., the number of sessions.

We consider several consistency models that are the most prevalent in practice. The weakest of them, *Read Committed* (RC) [11], requires that every value read in a transaction is written by a committed transaction. *Read Atomic* (RA) [16] requires that successive reads of the same variable in a transaction return the same value (also known as Repeatable Reads [11]), and that a transaction "sees" the values written by previous transactions in the same session. In general, we assume that transactions are organized in *sessions* [26], an abstraction of the sequence of transactions performed during the execution of an application. *Causal Consistency* (CC) [22] requires that if a transaction t_1 "affects" another transaction t_2, e.g., t_1 is ordered before t_2 in the same session or t_2 reads a value written by t_1, then these two transactions are observed by any other transaction in this order. *Prefix Consistency* (PC) [15] requires that there exists a total commit order between all the transactions such that each transaction observes a prefix of this sequence. *Snapshot Isolation* (SI) [11] further requires that two different transactions observe different prefixes if they both write to a common variable. Finally, we also provide new results concerning the problem of checking serializability (SER) that complement the known result about its NP-completeness.

The algorithmic issues we explore in this paper have led to a new specification framework for these consistency models that relies on the fact that the *write-read* relation in an execution (also known as *read-from*), relating reads with the transactions that wrote their value, can be defined effectively. The write-read relation can be extracted easily from executions where each value is written at most once (a variable can be written an arbitrary number of times). This can be easily enforced by tagging values with unique identifiers (e.g., a local counter that is incremented with every new write coupled with a client/session identifier)[1]. Since practical database implementations are data-independent [27], i.e., their behavior doesn't depend on the concrete values read or written in the transactions, any potential buggy behavior can be exposed in executions where each value is written at most once. Therefore, this assumption is without loss of generality.

Previous work [13, 14, 16] has formalized such consistency models using two auxiliary relations: a *visibility* relation defining for each transaction the set of transactions it observes, and a *commit order* defining the order in which transactions are committed to the "global" memory. An execution satisfying some consistency model is defined as the existence of a visibility relation and a commit order obeying certain axioms. In our case, the write-read relation derived from the execution plays the role of the visibility relation. This simplification allows us to state a series of axioms defining these consistency models, which have a common shape. Intuitively, they define lower bounds on the set of transactions t_1 that *must* precede in commit order a transaction t_2 that is read in the

[1] This is also used in Jepsen, e.g., checking dirty reads in Galera [2].

execution. Besides shedding a new light on the differences between these consistency models, these axioms are essential for the algorithmic issues we investigate afterwards.

We establish that checking whether an execution satisfies RC, RA, or CC is polynomial time, while the same problem is NP-complete for PC and SI. Moreover, in the case of the NP-complete consistency models (PC, SI, SER), we show that their verification problem becomes polynomial time provided that, roughly speaking, the number of sessions in the input executions is considered to be fixed (i.e., not counted for in the input size). In more detail, we establish that checking SER reduces to a search problem in a space that has polynomial size when the number of sessions is fixed. (This algorithm applies to arbitrary executions, but its complexity would be exponential in the number of sessions in general.) Then, we show that checking PC or SI can be reduced in polynomial time to checking SER using a transformation of executions that, roughly speaking, splits each transaction in two parts: one part containing all the reads, and one part containing all the writes (SI further requires adding some additional variables in order to deal with transactions writing on a common variable). We extend these results even further by relying on an abstraction of executions called *communication graphs* [17]. Roughly speaking, the vertices of a communication graph correspond to sessions, and the edges represent the fact that two sessions access (read or write) the same variable. We show that all these criteria are polynomial-time checkable provided that the *biconnected* components of the communication graph are of fixed size.

We provide an experimental evaluation of our algorithms on executions of CockroachDB [3], which claims to implement serializability [4] acknowledging however the possibility of anomalies, Galera [5], whose documentation contains contradicting claims about whether it implements snapshot isolation [6, 7], and AntidoteDB [8], which claims to implement causal consistency [9]. Our implementation reports violations of these criteria in all cases. The consistency violations we found for AntidoteDB are novel and have been confirmed by its developers. We show that our algorithms are efficient and scalable. In particular, we show that, although the asymptotic complexity of our algorithms is exponential in general (w.r.t. the number of sessions), the worst-case behavior is not exercised in practice.

To summarize, the contributions of this work are fourfold:

- We develop a new specification framework for describing common transactional-consistency criteria (§2);
- We show that checking RC, RA, and CC is polynomial time while checking PC and SI is NP-complete (§3);
- We show that PC, SI, and SER are polynomial-time checkable assuming that the communication graph of the input execution has fixed-size biconnected components (§4 and §5);
- We perform an empirical evaluation of our algorithms on executions generated by production databases (§6);

Combined, these contributions form an effective algorithmic framework for the verification of transactional-consistency models. To the best of our knowledge, we are the first to investigate the asymptotic complexity for most of these consistency models, despite their prevalence in practice.

Additional material can be found in [12].

2 CONSISTENCY CRITERIA

2.1 Histories

We consider a transactional database storing a set of variables $\mathsf{Var} = \{x, y, \ldots\}$. Clients interact with the database by issuing transactions formed of read and write operations. Assuming an unspecified

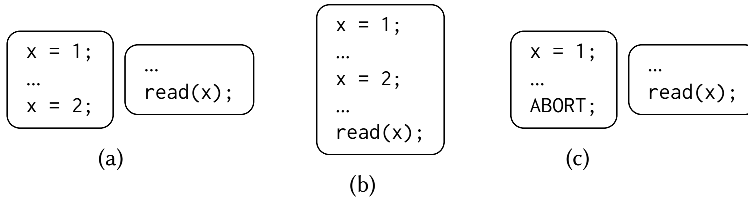


Fig. 1. Examples of transactions used to justify our simplifying assumptions (each box represents a different transaction): (a) only the last written value is observable in other transactions, (b) reads following writes to the same variable return the last written value in the same transaction, and (c) values written in aborted transactions are not observable.

set of values Val and a set of operation identifiers OpId, we let

$$\mathsf{Op} = \{\mathsf{read}_i(x, v), \mathsf{write}_i(x, v) : i \in \mathsf{OpId}, x \in \mathsf{Var}, v \in \mathsf{Val}\}$$

be the set of operations reading a value v or writing a value v to a variable x. We omit operation identifiers when they are not important.

Definition 2.1. A *transaction* $\langle O, \mathsf{po}\rangle$ is a finite set of operations O along with a strict total order po on O, called *program order*.

We use t, t_1, t_2, ... to range over transactions. The set of read, resp., write, operations in a transaction t is denoted by $\mathsf{reads}(t)$, resp., $\mathsf{writes}(t)$. The extension to sets of transactions is defined as usual. Also, we say that a transaction t *writes* a variable x, denoted by t writes x, when $\mathsf{write}_i(x, v) \in \mathsf{writes}(t)$ for some i and v. Similarly, a transaction t *reads* a variable x when $\mathsf{read}_i(x, v) \in \mathsf{reads}(t)$ for some i and v.

To simplify the exposition, we assume that each transaction t contains at most one write operation to each variable x [2], and that a read of a variable x cannot be preceded by a write to x in the same transaction[3]. If a transaction would contain multiple writes to the same variable, then only the last one should be visible to other transactions (w.r.t. any consistency criterion considered in practice). For instance, the `read(x)` in Figure 1a should not return 1 because this is not the last value written to `x` by the other transaction. It can return the initial value or 2. Also, if a read would be preceded by a write to the same variable in the same transaction, then it should return a value written in the same transaction (i.e., the value written by the latest write to x in that transaction). For instance, the `read(x)` in Figure 1b can only return 2 (assuming that there are no other writes on `x` in the same transaction). These two properties can be verified easily (in a syntactic manner) on a given execution. Beyond these two properties, the various consistency criteria used in practice constrain only the last writes to each variable in each transaction and the reads that are not preceded by writes to the same variable in the same transaction.

Consistency criteria are formalized on an abstract view of an execution called *history*. A history includes only successful or committed transactions. In the context of databases, it is always assumed that the effect of aborted transactions should not be visible to other transactions, and therefore, they can be ignored. For instance, the `read(x)` in Figure 1c should not return the value 1 written by the aborted transaction. The transactions are ordered according to a (partial) *session order* so which represents ordering constraints imposed by the applications using the database. Most often, so is a union of sequences, each sequence being called a *session*. We assume that the history includes a *write-read* relation that identifies the transaction writing the value returned by each read in the

[2]That is, for every transaction t, and every $\mathsf{write}(x, v)$, $\mathsf{write}(y, v') \in \mathsf{writes}(t)$, we have that $x \neq y$.
[3]That is, for every transaction $t = \langle O, \mathsf{po}\rangle$, if $\mathsf{write}(x, v) \in \mathsf{writes}(t)$ and there exists $\mathsf{read}(x, v) \in \mathsf{reads}(t)$, then we have that $\langle \mathsf{read}(x, v), \mathsf{write}(x, v)\rangle \in \mathsf{po}$

execution. As mentioned before, such a relation can be extracted easily from executions where each value is written at most once. Since in practice, databases are data-independent [27], i.e., their behavior does not depend on the concrete values read or written in the transactions, any potential buggy behavior can be exposed in such executions.

Definition 2.2. A *history* $\langle T, \mathsf{so}, \mathsf{wr}\rangle$ is a set of transactions T along with a strict partial order so called *session order*, and a relation $\mathsf{wr} \subseteq T \times \mathsf{reads}(T)$ called *write-read* relation, s.t.

- the inverse of wr is a total function, and if $(t, \mathsf{read}(x, v)) \in \mathsf{wr}$, then $\mathsf{write}(x, v) \in t$, and
- $\mathsf{so} \cup \mathsf{wr}$ is acyclic.

To simplify the technical exposition, we assume that every history includes a distinguished transaction writing the initial values of all variables. This transaction precedes all the other transactions in so. We use $h, h_1, h_2, \ldots$ to range over histories.

We say that the read operation $\mathsf{read}(x, v)$ reads value v from variable x written by t when $(t, \mathsf{read}(x, v)) \in \mathsf{wr}$. For a given variable x, wr_x denotes the restriction of wr to reads of variable x, i.e., , $\mathsf{wr}_x = \mathsf{wr} \cap (T \times \{\mathsf{read}(x, v) \mid v \in \mathsf{Val}\})$. Moreover, we extend the relations wr and wr_x to pairs of transactions as follows: $\langle t_1, t_2\rangle \in \mathsf{wr}$, resp., $\langle t_1, t_2\rangle \in \mathsf{wr}_x$, iff there exists a read operation $\mathsf{read}(x, v) \in \mathsf{reads}(t_2)$ such that $\langle t_1, \mathsf{read}(x, v)\rangle \in \mathsf{wr}$, resp., $\langle t_1, \mathsf{read}(x, v)\rangle \in \mathsf{wr}_x$. We say that the transaction t_1 is *read* by the transaction t_2 when $\langle t_1, t_2\rangle \in \mathsf{wr}$, and that it is *read* when it is read by some transaction t_2.

2.2 Axiomatic Framework

We describe an axiomatic framework to characterize the set of histories satisfying a certain consistency criterion. The overarching principle is to say that a history satisfies a certain criterion if there exists a strict total order on its transactions, called *commit order* and denoted by co, which extends the write-read relation and the session order, and which satisfies certain properties. These properties are expressed by a set of axioms that relate the commit order with the session-order and the write-read relation in the history.

The axioms we use have a uniform shape: they define mandatory co predecessors t_2 of a transaction t_1 that is read in the history. For instance, the criterion called Read Committed (RC) [11] requires that every value read in the history was written by a committed transaction, and also, that the reads in the same transaction are "monotonic" in the sense that they do not return values that are older, w.r.t. the commit order, than other values read in the past[4]. While the first condition holds for any history (because of the surjectivity of wr), the second condition is expressed by the axiom Read Committed in Figure 2a. This axiom states that for any transaction t_1 writing a variable x that is read in a transaction t, the set of transactions t_2 writing x and read previously in the same transaction must precede t_1 in commit order. For instance, Figure 3a shows a history and a (partial) commit order that does not satisfy this axiom because `read(x)` returns the value written in a transaction "older" than the transaction read in the previous `read(y)`. An example of a history and commit order satisfying this axiom is given in Figure 3b.

More precisely, the axioms are first-order formulas[5] of the following form:

$$\forall x,\ \forall t_1, t_2,\ \forall \alpha.\ t_1 \neq t_2 \land \langle t_1, \alpha\rangle \in \mathsf{wr}_x \land t_2 \text{ writes } x \land \phi(t_2, \alpha) \Rightarrow \langle t_2, t_1\rangle \in \mathsf{co}$$

where ϕ is a property relating t_2 and α (i.e., the read or the transaction reading from t_1) that varies from one axiom to another. Intuitively, this axiom schema states the following: in order for α to read specifically t_1's write on x, it must be the case that every t_2 that also writes x and satisfies

[4]This monotonicity property corresponds to the fact that in the original formulation of Read Committed [11], every write is guarded by the acquisition of a lock on the written variable, that is held until the end of the transaction.
[5]These formulas are interpreted on tuples $\langle h, \mathsf{co}\rangle$ of a history h and a commit order co on the transactions in h as usual.

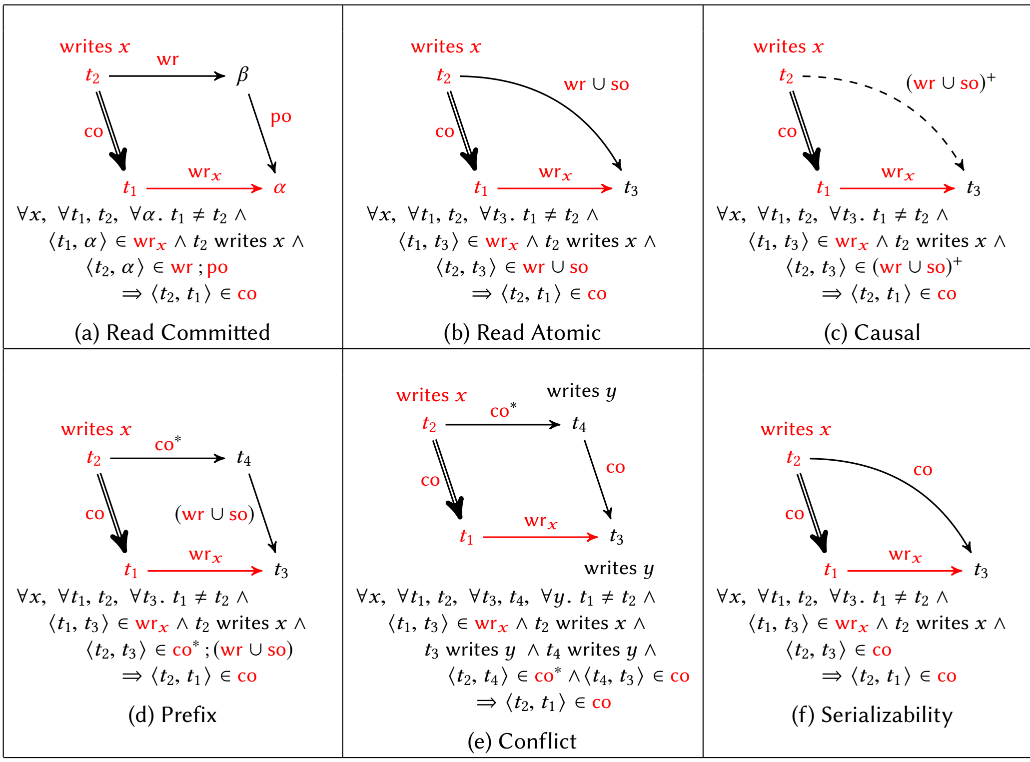
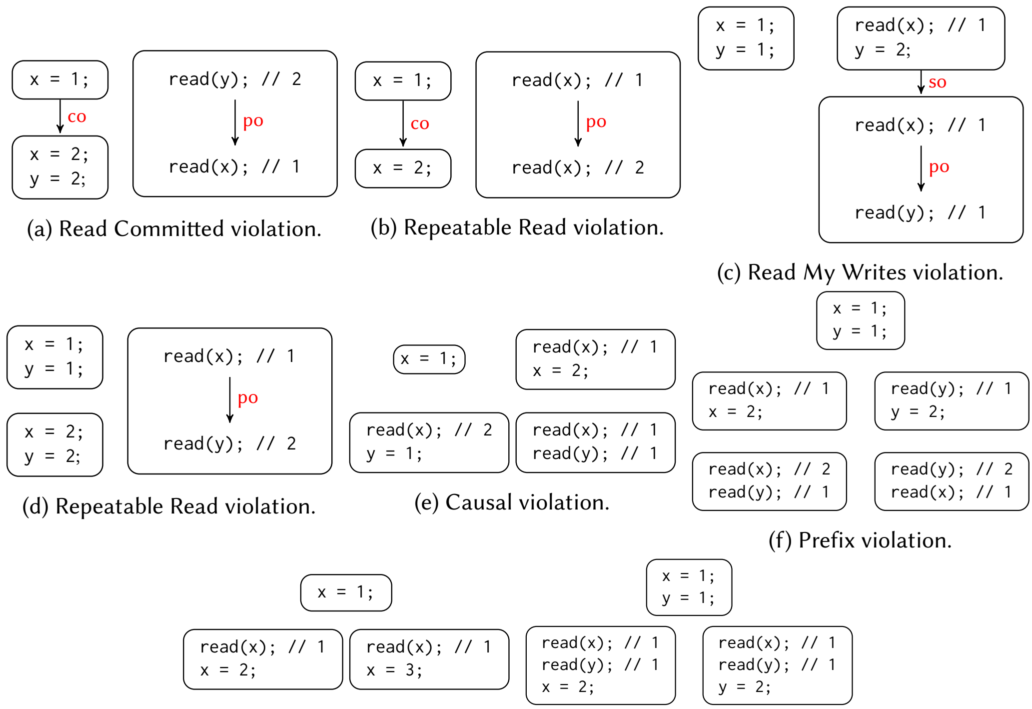


Fig. 2. Definitions of consistency axioms. The reflexive and transitive, resp., transitive, closure of a relation rel is denoted by rel^*, resp., rel^+. Also, ; denotes the composition of two relations, i.e., $rel_1 \,;\, rel_2 = \{\langle a, b\rangle | \exists c. \langle a, c\rangle \in rel_1 \wedge \langle c, b\rangle \in rel_2\}$.

$\phi(t_2, \alpha)$ was committed before t_1. Note that in all cases we consider, $\phi(t_2, \alpha)$ already ensures that t_2 is committed before the read α, so this axiom schema ensures that t_2 is furthermore committed before t_1's write.

The axioms used throughout the paper are given in Figure 2. The property ϕ relates t_2 and α using the write-read relation and the session order in the history, and the commit order.

In the following, we explain the rest of the consistency criteria we consider and the axioms defining them. Read Atomic (RA) [16] is a strengthening of Read Committed defined by the axiom Read Atomic, which states that for any transaction t_1 writing a variable x that is read in a transaction t_3, the set of wr or so predecessors of t_3 writing x must precede t_1 in commit order. The case of wr predecessors corresponds to the Repeatable Read criterion in [11] which requires that successive reads of the same variable in the same transaction return the same value, Figure 3b showing a violation, and also that every read of a variable x in a transaction t returns the value written by the maximal transaction t' (w.r.t. the commit order) that is read by t, Figure 3d showing a violation (for any commit order between the transactions on the left, either `read(x)` or `read(y)` will return a value not written by the maximal transaction). The case of so predecessors corresponds to the "read-my-writes" guarantee [26] concerning sessions, which states that a transaction t must observe previous writes in the same session. For instance, `read(y)` returning 1 in Figure 3c shows that the last transaction on the right does not satisfy this guarantee: the transaction writing 1 to `y` was already visible to that session before it wrote 2 to `y`, and therefore the value 2 should have



(a) Read Committed violation. (b) Repeatable Read violation. (c) Read My Writes violation. (d) Repeatable Read violation. (e) Causal violation. (f) Prefix violation. (g) Conflict violation. (h) Serializability violation.

Fig. 3. Examples of histories used to explain the axioms in Figure 2. For readability, the wr relation is defined by the values written in comments with each read.

Consistency model	Axioms
Read Committed (RC)	Read Committed
Read Atomic (RA)	Read Atomic
Causal consistency (CC)	Causal
Prefix consistency (PC)	Prefix
Snapshot isolation (SI)	Prefix $\wedge$ Conflict
Serializability (SER)	Serializability

Table 1. Consistency model definitions

been read. Read Atomic requires that the so predecessor of the transaction reading y be ordered in co before the transaction writing 1 to y, which makes the union $\mathsf{co} \cup \mathsf{wr}$ cyclic.

The following lemma shows that for histories satisfying Read Atomic, the inverse of wr_x extended to transactions is a total function.

Lemma 2.3. *Let $h = \langle T, \mathsf{so}, \mathsf{wr}\rangle$ be a history. If $\langle h, \mathsf{co}\rangle$ satisfies* Read Atomic*, then for every transaction t and two reads $read_{i_1}(x, v_1), read_{i_2}(x, v_2) \in \mathsf{reads}(t)$, $\mathsf{wr}^{-1}(read_{i_1}(x, v_1)) = \mathsf{wr}^{-1}(read_{i_2}(x, v_2))$ and $v_1 = v_2$.*

Causal Consistency (CC) [22] is defined by the axiom Causal, which states that for any transaction t_1 writing a variable x that is read in a transaction t_3, the set of $(\mathsf{wr} \cup \mathsf{so})^+$ predecessors of t_3 writing x must precede t_1 in commit order ($(\mathsf{wr} \cup \mathsf{so})^+$ is usually called the *causal* order). A violation of this axiom can be found in Figure 3e: the transaction t_2 writing 2 to x is a $(\mathsf{wr} \cup \mathsf{so})^+$

predecessor of the transaction t_3 reading 1 from x because the transaction t_4, writing 1 to y, reads x from t_2 and t_3 reads y from t_4. This implies that t_2 should precede in commit order the transaction t_1 writing 1 to x, which again, is inconsistent with the write-read relation (t_2 reads from t_1).

Prefix consistency (PC) [15] is a strengthening of CC, which requires that every transaction observes a prefix of a commit order between all the transactions. With the intuition that the observed transactions are $\mathsf{wr} \cup \mathsf{so}$ predecessors, the axiom Prefix defining PC, states that for any transaction t_1 writing a variable x that is read in a transaction t_3, the set of co^* predecessors of transactions observed by t_3 writing x must precede t_1 in commit order (we use co^* to say that even the transactions observed by t_3 must precede t_1). This ensures the prefix property stated above. An example of a PC violation can be found in Figure 3f: the two transactions on the bottom read from the three transactions on the top, but any serialization of those three transactions will imply that one of the combinations x=1, y=2 or x=2, y=1 cannot be produced at the end of a prefix in this serialization.

Snapshot Isolation (SI) [11] is a strengthening of PC that disallows two transactions to observe the same prefix of a commit order if they *conflict*, i.e., write to a common variable. It is defined by the conjunction of Prefix and another axiom called Conflict, which requires that for any transaction t_1 writing a variable x that is read in a transaction t_3, the set of co^* predecessors writing x of transactions conflicting with t_3 and before t_3 in commit order, must precede t_1 in commit order. Figure 3g shows a Conflict violation.

Finally, Serializability (SER) [25] is defined by the axiom with the same name, which requires that for any transaction t_1 writing to a variable x that is read in a transaction t_3, the set of co predecessors of t_3 writing x must precede t_1 in commit order. This ensures that each transaction observes the effects of all the co predecessors. Figure 3h shows a Serializability violation.

Lemma 2.4. *The following entailments hold:*

$$\mathsf{Causal} \Rightarrow \mathsf{Read\ Atomic} \Rightarrow \mathsf{Read\ Committed}$$
$$\mathsf{Prefix} \Rightarrow \mathsf{Causal}$$
$$\mathsf{Serializability} \Rightarrow \mathsf{Prefix} \wedge \mathsf{Conflict}$$

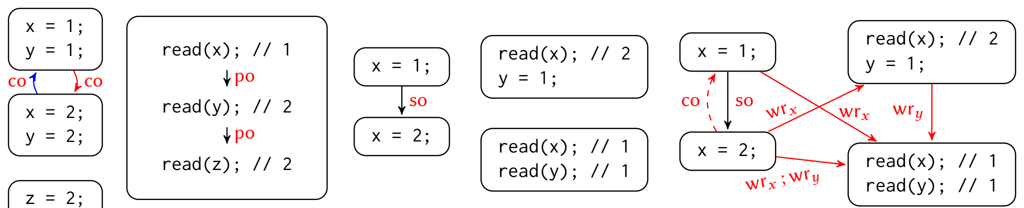
Definition 2.5. Given a set of axioms X defining a criterion C like in Table 1, a history $h = \langle T, \mathsf{so}, \mathsf{wr} \rangle$ *satisfies* C iff there exists a strict total order co such that $\mathsf{wr} \cup \mathsf{so} \subseteq \mathsf{co}$ and $\langle h, \mathsf{co} \rangle$ satisfies X.

Definition 2.5 and Lemma 2.4 imply that each consistency criterion in Table 1 is stronger than its predecessors (reading them from top to bottom), e.g., CC is stronger than RA and RC. This relation is strict, e.g., RA is not stronger than CC.

3 CHECKING CONSISTENCY CRITERIA

This section establishes the complexity of checking the different consistency criteria in Table 1 for a given history. More precisely, we show that Read Committed, Read Atomic, and Causal Consistency can be checked in polynomial time while the problem of checking the rest of the criteria is NP-complete.

Intuitively, the polynomial time results are based on the fact that the axioms defining those consistency criteria do not contain the commit order (co) on the left-hand side of the entailment. Therefore, proving the existence of a commit order satisfying those axioms can be done using a saturation procedure that builds a “partial” commit order based on instantiating the axioms on the write-read relation and the session order in the given history. Since the commit order must



(a) Violation of Read Atomic (b) Valid w.r.t. Read Atomic (c) Violation of Causal Consistency

Fig. 4. Applying the RA and CC checking algorithms.

Input: A history $h = \langle T, \mathsf{so}, \mathsf{wr} \rangle$
Output: *true* iff h satisfies Causal consistency

1 **if** $\mathsf{so} \cup \mathsf{wr}$ *is cyclic* **then**
2   **return** *false*;
3 $\mathsf{co} \leftarrow \mathsf{so} \cup \mathsf{wr}$;
4 **foreach** $x \in \mathsf{vars}(h)$ **do**
5   **foreach** $t_1 \neq t_2 \in T$ *s.t.* t_1 *and* t_2 *write* x **do**
6   **if** $\exists t_3.\ \langle t_1, t_3 \rangle \in \mathsf{wr}_x \wedge \langle t_2, t_3 \rangle \in (\mathsf{so} \cup \mathsf{wr})^+$ **then**
7             $\mathsf{co} \leftarrow \mathsf{co} \cup \{\langle t_2, t_1 \rangle\}$;
8 **if** co *is cyclic* **then**
9   **return** *false*;
10 **else**
11   **return** *true*;

Algorithm 1: Checking Causal consistency

be an extension of the write-read relation and the session order, it contains those two relations from the beginning. This saturation procedure stops when the order constraints derived this way become cyclic. For instance, let us consider applying such a procedure corresponding to RA on the histories in Figure 4a and Figure 4b. Applying the axiom in Figure 2b on the first history, since the transaction on the right reads 2 from y, we get that its wr_x predecessor (i.e., the first transaction on the left) must precede the transaction writing 2 to y in commit order (the red edge). This holds because the wr_x predecessor writes on y. Similarly, since the same transaction reads 1 from x, we get that its wr_y predecessor must precede the transaction writing 1 to x in commit order (the blue edge). This already implies a cyclic commit order, and therefore, this history does not satisfy RA. On the other hand, for the history in Figure 4b, all the axiom instantiations are vacuous, i.e., the left part of the entailment is false, and therefore, it satisfies RA. Checking CC on the history in Figure 4c requires a single saturation step: since the transaction on the bottom right reads 1 from x, its $\mathsf{wr}_x ; \mathsf{wr}_y$ predecessor that writes on x (the transaction on the bottom left) must precede in commit order the transaction writing 1 to x. Since this is already inconsistent with the session order, we get that this history violates CC.

Algorithm 1 lists our procedure for checking CC. As explained above, co is initially set to $\mathsf{so} \cup \mathsf{wr}$, and then, it is saturated with other ordering constraints implied by non-vacuous instantiations of the axiom Causal (where the left-hand side of the implication evaluates to true). The algorithms concerning RC and RA are defined in a similar way by essentially changing the test at line 6 so that it corresponds to the left-hand side of the implication in the corresponding axiom. Algorithm 1 can be rewritten as a Datalog program containing straightforward Datalog rules for computing transitive closures and relation composition, and a rule of the form[6]

$$\langle t_2, t_1 \rangle \in \mathsf{co} \text{ :- } t_1 \neq t_2, \langle t_1, t_3 \rangle \in \mathsf{wr}_x, \langle t_2, t_3 \rangle \in (\mathsf{so} \cup \mathsf{wr})^+$$

to represent the Causal axiom. The following is a consequence of the fact that these algorithms run in polynomial time (or equivalently, the Datalog programs can be evaluated in polynomial time over a database that contains the wr and so relations in a given history).

Theorem 3.1. *For any criterion $C \in \{$*Read Committed, Read Atomic, Causal consistency*$\}$, the problem of checking whether a given history satisfies C is polynomial time.*

On the other hand, checking PC, SI, and SER is NP-complete in general. We show this using a reduction from boolean satisfiability (SAT) that covers uniformly all the three cases. In the case of SER, it provides a new proof of the NP-completeness result by Papadimitriou [25] which uses a reduction from the so-called *non-circular* SAT and which cannot be extended to PC and SI.

Theorem 3.2. *For any criterion $C \in \{$*Prefix Consistency, Snapshot Isolation, Serializability*$\}$ the problem of checking whether a given history satisfies C is NP-complete.*

Proof. Given a history, any of these three criteria can be checked by guessing a total commit order on its transactions and verifying whether it satisfies the corresponding axioms. This shows that the problem is in NP.

To show NP-hardness, we define a reduction from boolean satisfiability. Therefore, let $\varphi = D_1 \wedge \ldots \wedge D_m$ be a CNF formula over the boolean variables $x_1, \ldots, x_n$ where each D_i is a disjunctive clause with m_i literals. Let λ_{ij} denote the j-th literal of D_i.

We construct a history h_φ such that φ is satisfiable if and only if h_φ satisfies PC, SI, or SER. Since $SER \Rightarrow SI \Rightarrow PC$, we show that (1) if h_φ satisfies PC, then φ is satisfiable, and (2) if φ is satisfiable, then h_φ satisfies SER.

The main idea of the construction is to represent truth values of each of the variables and literals in φ with the polarity of the commit order between corresponding transaction pairs. For each variable x_k, h_φ contains a pair of transactions a_k and b_k, and for each literal λ_{ij}, h_φ contains a set of transactions w_{ij}, y_{ij} and z_{ij} [7]. We want to have that x_k is false if and only if $\langle a_k, b_k \rangle \in \mathsf{co}$, and λ_{ij} is false if and only if $\langle y_{ij}, z_{ij} \rangle \in \mathsf{co}$ (the transaction w_{ij} is used to "synchronize" the truth value of the literals with that of the variables, which is explained later).

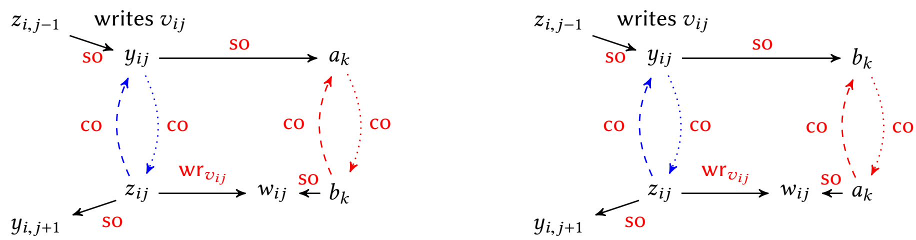
The history h_φ should ensure that the co ordering constraints corresponding to an assignment that falsifies the formula (i.e., one of its clauses) form a cycle. To achieve that, we add all pairs $\langle z_{ij}, y_{i,(j+1)\% m_i} \rangle$ in the session order so. An unsatisfied clause D_i, i.e., every λ_{ij} is false, leads to a cycle of the form $y_{i1} \xrightarrow{\mathsf{co}} z_{i1} \xrightarrow{\mathsf{so}} y_{i2} \xrightarrow{\mathsf{co}} z_{i2} \cdots z_{im_i} \xrightarrow{\mathsf{so}} y_{i1}$.

The most complicated part of the construction is to ensure the consistency between the truth value of the literals and the truth value of the variables, e.g., $\lambda_{ij} = x_k$ is false iff x_k is false. We use special sub-histories to enforce that if history h_φ satisfies PC (i.e., the axiom Prefix), then there exists a commit order co such that $\langle h_\varphi, \mathsf{co} \rangle$ satisfies Prefix (Figure 2d) and:

$$\begin{aligned} \langle a_k, b_k \rangle \in \mathsf{co} \text{ iff } \langle y_{ij}, z_{ij} \rangle \in \mathsf{co} \text{ when } \lambda_{ij} = x_k, \text{ and} \\ \langle a_k, b_k \rangle \in \mathsf{co} \text{ iff } \langle z_{ij}, y_{ij} \rangle \in \mathsf{co} \text{ when } \lambda_{ij} = \neg x_k. \end{aligned} \tag{1}$$

[6] We write Datalog rules using a standard notation *head* :- *body* where *head* is a relational atom (written as $\langle a, b \rangle \in R$ where a, b are elements and R a binary relation) and *body* is a list of relational atoms.

[7] We assume that the transactions a_k and b_k associated to a variable x_k are distinct and different from the transactions associated to another variable $x_{k'} \neq x_k$ or to a literal λ_{ij}. Similarly, for the transactions w_{ij}, y_{ij} and z_{ij} associated to a literal λ_{ij}.



(a) $\lambda_{ij} = x_k$ (b) $\lambda_{ij} = \neg x_k$

Fig. 5. Sub-histories included in h_φ for each literal λ_{ij} and variable x_k.

Figure 5a shows the sub-history associated to a positive literal $\lambda_{ij} = x_k$ while Figure 5b shows the case of a negative literal $\lambda_{ij} = \neg x_k$.

For a positive literal $\lambda_{ij} = x_k$ (Figure 5a), (1) we enrich session order with the pairs $\langle y_{ij}, a_k \rangle$ and $\langle b_k, w_{ij} \rangle$, (2) we include writes to a variable v_{ij} in the transactions y_{ij} and z_{ij}, and (3) we make w_{ij} read from z_{ij}, *i.e.*, $\langle z_{ij}, w_{ij} \rangle \in \mathsf{wr}_{v_{ij}}$. The case of a negative literal is similar, switching the roles of a_k and b_k.

This construction ensures that if the co goes downwards on the right-hand side ($\langle a_k, b_k \rangle \in \mathsf{co}$ in the case of a positive literal, and $\langle b_k, a_k \rangle \in \mathsf{co}$ in the case of a negative literal), then it must also go downwards on the left-hand side ($\langle y_{ij}, z_{ij} \rangle \in \mathsf{co}$) to satisfy Prefix. For instance, in the case of a positive literal, note that if $\langle a_k, b_k \rangle \in \mathsf{co}$, then $\langle y_{ij}, w_{ij} \rangle \in \mathsf{so} \,;\mathsf{co}\,; \mathsf{so}$. Therefore, for every commit order co such that $\langle h_\varphi, \mathsf{co} \rangle$ satisfies Prefix, $\langle a_k, b_k \rangle \in \mathsf{co}$ implies $\langle y_{ij}, z_{ij} \rangle \in \mathsf{co}$. Indeed, if $\langle a_k, b_k \rangle \in \mathsf{co}$, instantiating the Prefix axiom where y_{ij} plays the role of t_2, z_{ij} plays the role of t_1, and w_{ij} plays the role of t_3, we obtain that $\langle y_{ij}, z_{ij} \rangle \in \mathsf{co}$.

In contrast, when the co goes upwards on the right-hand side ($\langle b_k, a_k \rangle \in \mathsf{co}$ in the case of a positive literal, and $\langle a_k, b_k \rangle \in \mathsf{co}$ in the case of a negative literal) then it imposes no constraint on the direction of co on the left-hand side. Therefore, any commit order co satisfying Prefix that goes upwards on the right-hand side (e.g., $\langle b_k, a_k \rangle \in \mathsf{co}$ in the case of a positive literal) and downwards on the left-hand side ($\langle y_{ij}, z_{ij} \rangle \in \mathsf{co}$) in some sub-history (associated to some literal), thereby contradicting Property (1), can be modified into another commit order satisfying Prefix that goes upwards on the left-hand side as well. Formally, let co be a commit order such that $\langle h_\varphi, \mathsf{co} \rangle$ satisfies Prefix and

$$\langle b_k, a_k \rangle \in \mathsf{co} \land \langle y_{ij}, z_{ij} \rangle \in \mathsf{co}$$

for some literal $\lambda_{ij} = x_k$ (the case of negative literals can be handled in a similar manner). Let co_1 be the restriction of co on the set of tuples

$$\{\langle a_{k'}, b_{k'} \rangle, \langle b_{k'}, a_{k'} \rangle | 1 \leq k' \leq n\} \cup \{\langle y_{i'j'}, z_{i'j'} \rangle, \langle z_{i'j'}, y_{i'j'} \rangle | \text{for each } i', j'\} \cup \mathsf{so} \cup \mathsf{wr}.$$

Since $\mathsf{co}_1 \subseteq \mathsf{co}$, we have that co_1 is acyclic. Let co_2 be a relation obtained from co_1 by flipping the order between y_{ij} and z_{ij} (i.e., $\mathsf{co}_2 = \mathsf{co}_1 \setminus \{\langle y_{ij}, z_{ij} \rangle\} \cup \{\langle z_{ij}, y_{ij} \rangle\}$). This flipping does not introduce any cycle because co_2 contains no path ending in z_{ij} (see Fig 5a). Also, co_2 still satisfies the Prefix axiom (since $\langle b_k, a_k \rangle \in \mathsf{co}_2$ there is no path from y_{ij} to w_{ij} satisfying the constraints in the Prefix axiom). Since co_2 is acyclic, it can be extended to a total commit order co_3 that satisfies Prefix. This is a consequence of the following lemma whose proof follows easily from definitions (the part of this lemma concerning Serializability will be used later).

Lemma 3.3. *Let co be an acyclic relation that includes* so $\cup$ wr, $\langle a_k, b_k\rangle$ *or* $\langle b_k, a_k\rangle$, *for each k, and* $\langle y_{ij}, z_{ij}\rangle$ *or* $\langle z_{ij}, y_{ij}\rangle$, *for each i, j. For each axiom* $A \in \{\text{Prefix}, \text{Serializability}\}$, *if* $\langle h_\varphi, \text{co}\rangle$ *satisfies A, then there exists a total commit order* co′ *such that* co $\subseteq$ co′ *and* $\langle h_\varphi, \text{co}'\rangle$ *satisfies A.*

Therefore, $\langle h_\varphi, \text{co}_3\rangle$ satisfies Prefix, and $\langle b_k, a_k\rangle \in \text{co}_3 \wedge \langle z_{ij}, y_{ij}\rangle \in \text{co}_3$ (co_3 goes upwards on both sides of a sub-history like in Figure 5a). This transformation can be applied iteratively until obtaining a commit order that satisfies both Prefix and Property (1).

Next, we complete the correctness proof of this reduction. For the "if" direction, if h_φ satisfies PC, then there exists a total commit order co between the transactions described above, which together with h_φ satisfies Prefix. The assignment of the variables x_k explained above (defined by the co order between a_k and b_k, for each k) satisfies the formula φ since there exists no cycle between the transactions y_{ij} and z_{ij}, which implies that for each clause D_i, there exists a j such that $\langle y_{ij}, z_{ij}\rangle \notin \text{co}$ which means that λ_{ij} is satisfied. For the "only-if" direction, let γ be a satisfying assignment for φ. Also, let co′ be a binary relation that includes so and wr such that if $\gamma(x_k) = \mathit{false}$, then $\langle a_k, b_k\rangle \in \text{co}'$, $\langle y_{ij}, z_{ij}\rangle \in \text{co}'$ for each $\lambda_{ij} = x_k$, and $\langle z_{ij}, y_{ij}\rangle \in \text{co}'$ for each $\lambda_{ij} = \neg x_k$, and if $\gamma(x_k) = \mathit{true}$, then $\langle b_k, a_k\rangle \in \text{co}'$, $\langle z_{ij}, y_{ij}\rangle \in \text{co}'$ for each $\lambda_{ij} = x_k$, and $\langle y_{ij}, z_{ij}\rangle \in \text{co}'$ for each $\lambda_{ij} = \neg x_k$. Note that co′ is acyclic: no cycle can contain w_{ij} because w_{ij} has no "outgoing" dependency (i.e., co′ contains no pair with w_{ij} as a first component), there is no cycle including some pair of transactions a_k, b_k and some pair y_{ij}, z_{ij} because there is no way to reach y_{ij} or z_{ij} from a_k or b_k, there is no cycle including only transactions a_k and b_k because a_{k_1} and b_{k_1} are not related to a_{k_2} and b_{k_2}, for $k_1 \neq k_2$, there is no cycle including transactions y_{i_1,j_1}, z_{i_1,j_1} and y_{i_2,j_2}, z_{i_2,j_2} for $i_1 \neq i_2$ since these are disconnected as well, and finally, there is no cycle including only transactions y_{ij} and z_{ij}, for a fixed i, because φ is satisfiable. By Lemma 3.3, the acyclic relation co′ can be extended to a total commit order co which together with h_φ satisfies the Serializability axiom. Therefore, h_φ satisfies SER. □

4 CHECKING CONSISTENCY OF BOUNDED-WIDTH HISTORIES

In this section, we show that checking prefix consistency, snapshot isolation, and serializability becomes polynomial time under the assumption that the *width* of the given history, i.e., the maximum number of mutually-unordered transactions w.r.t. the session order, is bounded by a fixed constant. If we consider the standard case where the session order is a union of transaction sequences (modulo the fictitious transaction writing the initial values), i.e., a set of sessions, then the width of the history is the number of sessions. We start by presenting an algorithm for checking serializability that is polynomial time when the width is bounded by a fixed constant. In general, the asymptotic complexity of this algorithm is exponential in the width of the history, but this worst-case behavior is not exercised in practice as shown in Section 6. Then, we prove that checking prefix consistency and snapshot isolation can be reduced in polynomial time to the problem of checking serializability.

4.1 Checking Serializability

We present an algorithm for checking serializability of a given history which constructs a valid commit order (satisfying Serialization), if any, by "linearizing" transactions one by one in an order consistent with the session order. At any time, the set of already linearized transactions is uniquely determined by an antichain of the session order (i.e., a set of mutually-unordered transactions w.r.t. so), and the next transaction to linearize is chosen among the immediate so successors of the transactions in this antichain. The crux of the algorithm is that the next transaction to linearize can be chosen such that it does not produce violations of Serialization in a way that does not depend on the order between the already linearized transactions. Therefore, the algorithm can be seen as a

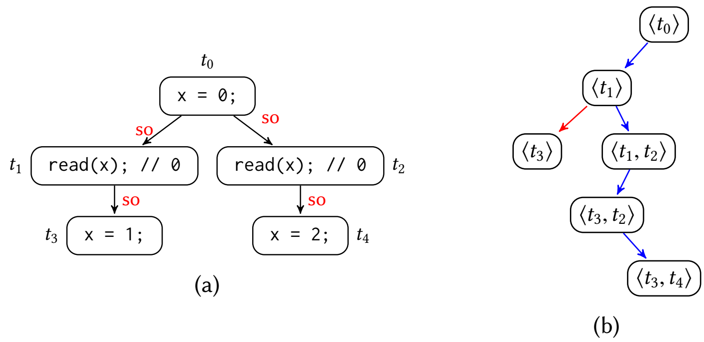


Fig. 6. Applying the serializability checking algorithm checkSER (Algorithm 2) on the serializable history on the left. The right part pictures a search for valid extensions of serializable prefixes, represented by their boundaries. The red arrow means that the search is blocked (the prefix at the target is not a valid extension), while blue arrows mean that the search continues.

search in the space of so antichains. If the width of the history is bounded (by a fixed constant), then the number of possible so antichains is polynomial in the size of the history, which implies that the search can be done in polynomial time.

A *prefix* of a history $h = \langle T, \mathsf{so}, \mathsf{wr} \rangle$ is a set of transactions $T' \subseteq T$ such that all the so predecessors of transactions in T' are also in T', i.e., $\forall t \in T.\ \mathsf{so}^{-1}(t) \in T$. A prefix T' is uniquely determined by the set of transactions in T' that are maximal w.r.t. so. This set of transactions forms an *antichain* of so, i.e., any two elements in this set are incomparable w.r.t. so. Given an antichain $\{t_1, \ldots, t_n\}$ of so, we say that $\{t_1, \ldots, t_n\}$ is the *boundary* of the prefix $T' = \{t : \exists i.\ \langle t, t_i \rangle \in \mathsf{so} \lor t = t_i\}$. For instance, given the history in Figure 6a, the set of transactions $\{t_0, t_1, t_2\}$ is a prefix with boundary $\{t_1, t_2\}$ (the latter is an antichain of the session order).

A prefix T' of a history h is called *serializable* iff there exists a *partial* commit order co on the transactions in h such that the following hold:

- co does not contradict the session order and the write-read relation in h, i.e., $\mathsf{wr} \cup \mathsf{so} \cup \mathsf{co}$ is acyclic,
- co is a total order on transactions in T',
- co orders transactions in T' before transactions in $T \setminus T'$, i.e., $\langle t_1, t_2 \rangle \in \mathsf{co}$ for every $t_1 \in T'$ and $t_2 \in T \setminus T'$,
- co does not order any two transactions $t_1, t_2 \notin T'$
- the history h along with the commit order co satisfies the axiom defining serializability, i.e., $\langle h, \mathsf{co} \rangle \models$ Serialization.

For the history in Figure 6a, the prefix $\{t_0, t_1, t_2\}$ is serializable since there exists a partial commit order co that orders t_0, t_1, t_2 in this order, and both t_1 and t_2 before t_3 and t_4. The axiom Serialization is satisfied trivially, since the prefix contains a single transaction writing x and all the transactions outside of the prefix do not read x.

A prefix $T' \uplus \{t\}$ of h is called a *valid extension* of a serializable prefix T' of h [8], denoted by $T' \rhd T' \uplus \{t\}$if:

- t does not read from a transaction outside of T', i.e., for every $t' \in T \setminus T'$, $\langle t', t \rangle \notin \mathsf{wr}$, and
- for every variable x written by t, there exists no transaction $t_2 \neq t$ outside of T' that reads a value of x written by a transaction t_1 in T', i.e., for every x written by t and every $t_1 \in T'$ and $t_2 \in T \setminus (T' \uplus \{t\})$, $\langle t_1, t_2 \rangle \notin \mathsf{wr}$.

[8]We assume that $t \notin T'$ which is implied by the use of the disjoint union $\uplus$.

Input: A history $h = (T, \mathsf{so}, \mathsf{wr})$, a serializable prefix T' of h
Output: *true* iff $T' \rhd^* h$

if $T' = T$ **then**
  return *true*;
foreach $t \notin T'$ *s.t.* $\forall t' \notin T'.\ \langle t', t\rangle \notin \mathsf{wr} \cup \mathsf{so}$ **do**
  if $T' \not\rhd T' \uplus \{t\}$ **then**
    continue;
  if $T' \uplus \{t\} \notin \mathit{seen} \land \mathsf{checkSER}(h, T' \uplus \{t\})$ **then**
    return *true*;
  $\mathit{seen} \leftarrow \mathit{seen} \cup \{(T' \uplus \{t\})\}$;
return *false*;

Algorithm 2: The algorithm checkSER for checking serializabilty. *seen* is a global variable storing a set of prefixes of h (which are not serializable). It is initialized as the empty set.

For the history in Figure 6a, we have $\{t_0, t_1\} \rhd \{t_0, t_1\} \uplus \{t_2\}$ because t_2 reads from t_0 and it does not write any variable. On the other hand $\{t_0, t_1\} \not\rhd \{t_0, t_1\} \uplus \{t_3\}$ because t_3 writes x and the transaction t_2, outside of this prefix, reads from the transaction t_0 included in the prefix.

Let $\rhd^*$ denote the reflexive and transitive closure of $\rhd$.

The following lemma is essential in proving that iterative valid extensions of the initial empty prefix can be used to show that a given history is serializable.

Lemma 4.1. *For a serializable prefix T' of a history h, a prefix $T' \uplus \{t\}$ is serializable if it is a valid extension of T'.*

Proof. Let co' be the partial commit order for T' which satisfies the serializable prefix conditions. We extend co' to a partial order $\mathsf{co} = \mathsf{co}' \cup \{\langle t, t'\rangle | t' \notin T' \uplus \{t'\}\}$. We show that $\langle h, \mathsf{co}\rangle \models$ Serialization. The other conditions for $T' \uplus \{t\}$ being a serializable prefix are satisfied trivially by co.

Assume by contradiction that $\langle h, \mathsf{co}\rangle$ does not satisfy the axiom Serialization. Then, there exists $t_1, t_2, t_3, x \in \mathsf{vars}(h)$ s.t. $\langle t_1, t_3\rangle \in \mathsf{wr}_x$ and t_2 writes on x and $\langle t_1, t_2\rangle, \langle t_2, t_3\rangle \in \mathsf{co}$. Since $\langle h, \mathsf{co}'\rangle$ satisfies this axiom, at least one of these two co ordering constraints are of the form $\langle t, t'\rangle$ where $t' \notin T' \uplus \{t\}$:

- the case $t_1 = t$ and $t_2 \notin T' \uplus \{t\}$ is not possible because co' contains no pair of the form $\langle t', \_\rangle \in \mathsf{co}'$ with $t' \notin T'$ (recall that $\langle t_2, t_3\rangle$ should be also included in co).
- If $t_2 = t$ then, $\langle t_1, t_2\rangle \in \mathsf{co}'$ and $\langle t_2, t_3\rangle$ for some $t_3 \notin T' \uplus \{t\}$. But, by the definition of valid extension, for all variables x written by t, there exists no transaction $t_3 \notin T' \uplus \{t\}$ such that it reads x from $t_1 \in T'$. Therefore, this is also a contradiction. □

Algorithm 2 lists our algorithm for checking serializability. It is defined as a recursive procedure that searches for a sequence of valid extensions of a given prefix (initially, this prefix is empty) until covering the whole history. Figure 6b pictures this search on the history in Figure 6a. The right branch (containing blue edges) contains only valid extensions and it reaches a prefix that includes all the transactions in the history.

Theorem 4.2. *A history h is serializable iff* $\mathsf{checkSER}(h, \emptyset)$ *returns true.*

Proof. The "if" direction is a direct consequence of Lemma 4.1. For the reverse, assume that $h = \langle T, \mathsf{so}, \mathsf{wr}\rangle$ is serializable with a (total) commit order co. Let co_i be the set of transactions in the prefix of co of length i. Since co is consistent with so, we have that co_i is a prefix of h, for any i. We show by induction that co_{i+1} is a valid extension of co_i. The base case is trivial. For the induction step, let t be the last transaction in the prefix of co of length $i+1$. Then,

- t cannot read from a transaction outside of co_i because co is consistent with the write-read relation wr,
- also, for every variable x written by t, there exists no transaction $t_2 \neq t$ outside of co_i that reads a value of x written by a transaction $t_1 \in \mathsf{co}_i$. Otherwise, $\langle t_1, t_2 \rangle \in \mathsf{wr}_x$, $\langle t, t_2 \rangle \in \mathsf{co}$, and $\langle t_1, t \rangle \in \mathsf{co}$ which implies that $\langle h, \mathsf{co} \rangle$ does not satisfy Serializability.

This implies that $\mathsf{checkSER}(h, \emptyset)$ returns true. □

Algorithm 2 enumerates prefixes of the given history h, each prefix being uniquely determined by an antichain of h containing the so-maximal transactions in that prefix. By definition, the size of each antichain of a history h is smaller than the width of h. Therefore, the number of possible antichains (prefixes) of a history h is $O(\mathsf{size}(h)^{\mathsf{width}(h)})$ where $\mathsf{size}(h)$, resp., $\mathsf{width}(h)$, is the number of transactions, resp., the width, of h. Since the valid extension property can be checked in quadratic time, the asymptotic time complexity of the algorithm defined by checkSER is upper bounded by $O(\mathsf{size}(h)^{\mathsf{width}(h)} \cdot \mathsf{size}(h)^3)$. The following corollary is a direct consequence of these observations.

Corollary 4.3. *For an arbitrary but fixed constant $k \in \mathbb{N}$, the problem of checking serializability for histories of width at most k is polynomial time.*

4.2 Reducing Prefix Consistency to Serializability

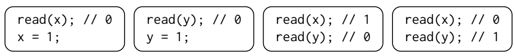
We describe a polynomial time reduction of checking prefix consistency of bounded-width histories to the analogous problem for serializability. Intuitively, as opposed to serializability, prefix consistency allows that two transactions read the same snapshot of the database and commit together even if they write on the same variable. Based on this observation, given a history h for which we want to check prefix consistency, we define a new history $h_{R|W}$ where each transaction t is split into a transaction performing all the reads in t and another transaction performing all the writes in t (the history $h_{R|W}$ retains all the session order and write-read dependencies of h). We show that if the set of read and write transactions obtained this way can be shown to be serializable, then the original history satisfies prefix consistency, and vice-versa. For instance, Figure 7 shows this transformation on the two histories in Figure 7a and Figure 7c, which represent typical anomalies known as "long fork" and "lost update", respectively. The former is not admitted by PC while the latter is admitted. It can be easily seen that the transformed history corresponding to the "long fork" anomaly is not serializable while the one corresponding to "lost update" is serializable. We show that this transformation leads to a history of the same width, which by Corollary 4.3, implies that checking prefix consistency of bounded-width histories is polynomial time.

Thus, given a history $h = \langle T, \mathsf{wr}, \mathsf{so} \rangle$, we define the history $h_{R|W} = \langle T', \mathsf{wr}', \mathsf{so}' \rangle$ as follows:

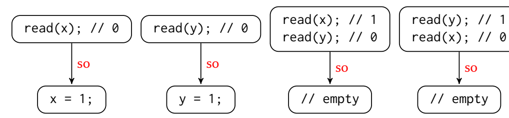
- T' contains a transaction R_t, called a *read* transaction, and a transaction W_t, called a *write* transaction, for each transaction t in the original history, i.e., $T' = \{R_t | t \in T\} \cup \{W_t | t \in T\}$
- the write transaction W_t writes exactly the same set of variables as t, i.e., for each variable x, W_t writes to x iff t writes to x.
- the read transaction R_t reads exactly the same values and the same variables as t, i.e., for each variable x, $\mathsf{wr}_x' = \{\langle W_{t_1}, R_{t_2} \rangle | \langle t_1, t_2 \rangle \in \mathsf{wr}_x\}$
- the session order between the read and the write transactions corresponds to that of the original transactions and read transactions precede their write counterparts, i.e.,

$$\mathsf{so}' = \{\langle R_t, W_t \rangle | t \in T\} \cup \{\langle R_{t_1}, R_{t_2} \rangle, \langle R_{t_1}, W_{t_2} \rangle, \langle W_{t_1}, R_{t_2} \rangle, \langle W_{t_1}, W_{t_2} \rangle | \langle t_1, t_2 \rangle \in \mathsf{so}\}$$

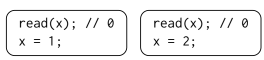
The following lemma is a straightforward consequence of the definitions.



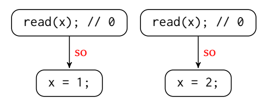
(a) Long fork



(b) Long fork (transformed)



(c) Lost update



(d) Lost update (transformed)

Fig. 7. Reducing PC to SER. Initially, the value of every variable is 0.

Lemma 4.4. *The histories h and $h_{R|W}$ have the same width.*

Next, we show that $h_{R|W}$ is serializable if h is prefix consistent. Formally, we show that

$$\forall \mathsf{co}.\ \exists \mathsf{co}'.\ \langle h, \mathsf{co}\rangle \models \mathsf{Prefix} \Rightarrow \left\langle h_{R|W}, \mathsf{co}'\right\rangle \models \mathsf{Serializability}$$

Thus, let co be a commit (total) order on transactions of h which together with h satisfies the prefix consistency axiom. We define two *partial* commit orders co'_1 and co'_2, co'_2 a strengthening of co'_1, which we prove that they are acyclic and that any linearization co' of co'_2 is a valid witness for $h_{R|W}$ satisfying serializability.

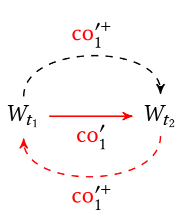
Thus, let co'_1 be a *partial* commit order on transactions of $h_{R|W}$ defined as follows:

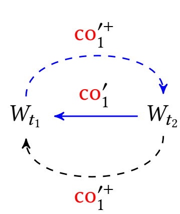
$$\mathsf{co}'_1 = \{\langle R_t, W_t\rangle | t \in T\} \cup \{\left\langle W_{t_1}, W_{t_2}\right\rangle | \langle t_1, t_2\rangle \in \mathsf{co}\} \ \cup \{\left\langle W_{t_1}, R_{t_2}\right\rangle | \langle t_1, t_2\rangle \in \mathsf{wr} \cup \mathsf{so}\}$$

We show that if co'_1 were to be cyclic, then it contains a minimal cycle with one read transaction, and at least one but at most two write transactions. Then, we show that such cycles cannot exist.

Lemma 4.5. *The relation co'_1 is acyclic.*

Proof. We first show that if co'_1 were to be cyclic, then it contains a minimal cycle with one read transaction, and at least one but at most two write transactions. Then, we show that such cycles cannot exist. Therefore, let us assume that co'_1 is cyclic. Then,





(a) $\left\langle W_{t_1}, W_{t_2}\right\rangle \in \mathsf{co}'_1$ (b) $\left\langle W_{t_2}, W_{t_1}\right\rangle \in \mathsf{co}'_1$

Fig. 8. Cycles with non-consecutive write transactions.

- Since $\left\langle W_{t_1}, W_{t_2}\right\rangle \in \mathsf{co}'_1$ implies $\langle t_1, t_2\rangle \in \mathsf{co}$, for every t_1 and t_2, a cycle in co'_1 cannot contain only write transactions. Otherwise, it will imply a cycle in the original commit order co. Therefore, a cycle in co'_1 must contain at least one read transaction.
- Assume that a cycle in co'_1 contains two write transactions W_{t_1} and W_{t_2} which are not consecutive, like in Figure 8. Since either $\left\langle W_{t_1}, W_{t_2}\right\rangle \in \mathsf{co}'_1$ or $\left\langle W_{t_1}, W_{t_2}\right\rangle \in \mathsf{co}'_1$, there exists a smaller cycle in co'_1 where these two write transactions are consecutive. If $\left\langle W_{t_1}, W_{t_2}\right\rangle \in \mathsf{co}'_1$, then co'_1 contains the smaller cycle on the lower part of the original cycle (Figure 8a), and if $\left\langle W_{t_2}, W_{t_1}\right\rangle \in \mathsf{co}'_1$, then co'_1 contains the cycle on the upper part of the original cycle (Figure 8b). Thus, all the write transactions in a minimal cycle of co'_1 must be consecutive.
- If a minimal cycle were to contain three write transactions, then all of them cannot be consecutive unless they all three form a cycle, which is not possible. So a minimal cycle contains at most two write transactions.

- Since co'_1 contains no direct relation between read transactions, it cannot contain a cycle with two consecutive read transactions, or only read transactions.

This shows that a minimal cycle of co'_1 would include a read transaction and a write transaction, and at most one more write transaction. We prove that such cycles are however impossible:

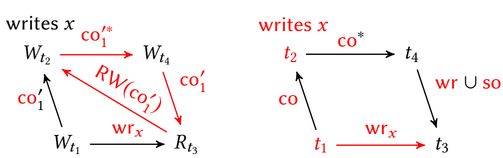
- if the cycle is of size 2, then it contains two transactions W_{t_1} and R_{t_2} such that $\langle W_{t_1}, R_{t_2}\rangle \in \mathsf{co}'_1$ and $\langle R_{t_2}, W_{t_1}\rangle \in \mathsf{co}'_1$. Since all the $\langle R\_, W\_\rangle$ dependencies in co'_1 are of the form $\langle R_t, W_t\rangle$, it follows that $t_1 = t_2$. Then, we have $\langle W_{t_1}, R_{t_1}\rangle \in \mathsf{co}'_1$ which implies $\langle t_1, t_1\rangle \in \mathsf{wr} \cup \mathsf{so}$, a contradiction.
- if the cycle is of size 3, then it contains three transactions W_{t_1}, W_{t_2}, and R_{t_3} such that $\langle W_{t_1}, W_{t_2}\rangle \in \mathsf{co}'_1$, $\langle W_{t_2}, R_{t_3}\rangle \in \mathsf{co}'_1$, and $\langle R_{t_3}, W_{t_1}\rangle \in \mathsf{co}'_1$. Using a similar argument as in the previous case, $\langle R_{t_3}, W_{t_1}\rangle \in \mathsf{co}'_1$ implies $t_3 = t_1$. Therefore, $\langle t_1, t_2\rangle \in \mathsf{co}$ and $\langle t_2, t_1\rangle \in \mathsf{wr} \cup \mathsf{so}$, which contradicts the fact that $\mathsf{wr} \cup \mathsf{so} \subseteq \mathsf{co}$. □

We define a strengthening of co'_1 where intuitively, we add all the dependencies from read transactions t_3 to write transactions t_2 that "overwrite" values read by t_3. Formally, $\mathsf{co}'_2 = \mathsf{co}'_1 \cup \mathsf{RW}(\mathsf{co}'_1)$ where

$$\mathsf{RW}(\mathsf{co}'_1) = \{\langle t_3, t_2\rangle | \exists x \in \mathsf{vars}(h).\ \exists t_1 \in T'.\ \langle t_1, t_3\rangle \in \mathsf{wr}_x{}', \langle t_1, t_2\rangle \in \mathsf{co}'_1, t_2 \text{ writes } x\}$$

It can be shown that any cycle in co'_2 would correspond to a Prefix violation in the original history. Therefore,

Lemma 4.6. *The relation co'_2 is acyclic.*



(a) Minimal cycle in co'_2. (b) Prefix violation in $\langle h, \mathsf{co}\rangle$.

Fig. 9. Cycles in co'_2 correspond to Prefix violations.

Proof. Assume that co'_2 is cyclic. Any minimal cycle in co'_2 still satisfies the properties of minimal cycles of co'_1 proved in Lemma 4.5 (because all write transactions are still totally ordered and co'_2 doesn't relate directly read transactions). So, a minimal cycle in co'_2 contains a read transaction and a write transaction, and at most one more write transaction.

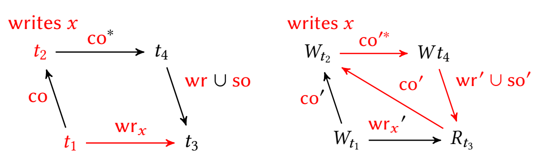
Since co'_1 is acyclic, a cycle in co'_2, and in particular a minimal one, must necessarily contain a dependency from $\mathsf{RW}(\mathsf{co}'_1)$. Note that a minimal cycle cannot contain two such dependencies since this would imply that it contains two non-consecutive write transactions. The red edges in Figure 9a show a minimal cycle of co'_2 satisfying all the properties mentioned above. This cycle contains a dependency $\langle R_{t_3}, W_{t_2}\rangle \in \mathsf{RW}(\mathsf{co}'_1)$ which implies the existence of a write transaction W_{t_1} in $h_{R|W}$ s.t. $\langle W_{t_1}, R_{t_3}\rangle \in \mathsf{wr}_x{}'$ and $\langle W_{t_1}, W_{t_2}\rangle \in \mathsf{co}'_1$ and W_{t_1}, W_{t_2} write on x (these dependencies are represented by the black edges in Figure 9a). The relations between these transactions of $h_{R|W}$ imply that the corresponding transactions of h are related as shown in Figure 9b: $\langle W_{t_1}, W_{t_2}\rangle \in \mathsf{co}'_1$ and $\langle W_{t_2}, W_{t_4}\rangle \in \mathsf{co}'^*_1$ imply $\langle t_1, t_2\rangle \in \mathsf{co}$ and $\langle t_2, t_4\rangle \in \mathsf{co}^*$, respectively, $\langle W_{t_1}, W_{t_3}\rangle \in \mathsf{wr}_x{}'$ implies $\langle t_1, t_3\rangle \in \mathsf{wr}_x$, and $\langle W_{t_4}, R_{t_3}\rangle \in \mathsf{co}'_1$ implies $\langle t_4, t_3\rangle \in \mathsf{wr} \cup \mathsf{so}$. This implies that $\langle h, \mathsf{co}\rangle$ doesn't satisfy the Prefix axiom, a contradiction. □

Lemma 4.7. *If a history h satisfies prefix consistency, then $h_{R|W}$ is serializable.*

Proof. Let co' be any total order consistent with co'_2. Assume by contradiction that $\langle h_{R|W}, \mathsf{co}'\rangle$ doesn't satisfy Serializability. Then, there exist $t'_1, t'_2, t'_3 \in T'$ such that $\langle t'_1, t'_2\rangle, \langle t'_2, t'_3\rangle \in \mathsf{co}'$ and t'_1, t'_2 write on some variable x and $\langle t'_1, t'_3\rangle \in \mathsf{wr}_x{}'$. But then t'_1, t'_2 are write transactions and co'_1 must contain $\langle t'_1, t'_2\rangle$. Therefore, $\mathsf{RW}(\mathsf{co}'_1)$ and co'_2 should contain $\langle t'_3, t'_2\rangle$, a contradiction with co' being consistent with co'_2. □

Finally, it can be proved that any linearization co' of co'_2 satisfies Serializability (together with $h_{R|W}$). Moreover, it can also be shown that the serializability of $h_{R|W}$ implies that h satisfies PC. Therefore,

Theorem 4.8. *A history h satisfies prefix consistency iff $h_{R|W}$ is serializable.*



(a) Prefix violation in $\langle h, \mathsf{co}\rangle$ (b) Cycle in co'.

Fig. 10. Prefix violations correspond to cycles in co'.

Proof. The "only-if" direction is proven by Lemma 4.7. For the reverse, we show that $\forall \mathsf{co}'. \exists \mathsf{co}. \left\langle h_{R|W}, \mathsf{co}'\right\rangle \models \text{Serializability} \Rightarrow \langle h, \mathsf{co}\rangle \models \text{Prefix}$

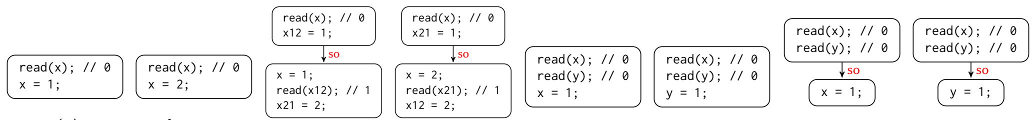
Thus, let co' be a commit (total) order on transactions of $h_{R|W}$ which together with $h_{R|W}$ satisfies the serializability axiom. Let co be a commit order on transactions of h defined by $\mathsf{co} = \{\langle t_1, t_2\rangle | \left\langle W_{t_1}, W_{t_2}\right\rangle \in \mathsf{co}'\}$ (co is clearly a total order). If co were not to be consistent with $\mathsf{wr} \cup \mathsf{so}$, then there would exist transactions t_1 and t_2 such that $\langle t_1, t_2\rangle \in \mathsf{wr} \cup \mathsf{so}$ and $\langle t_2, t_1\rangle \in \mathsf{co}$, which would imply that $\left\langle W_{t_1}, R_{t_2}\right\rangle, \left\langle R_{t_2}, W_{t_2}\right\rangle \in \mathsf{wr} \cup \mathsf{so}$ and $\left\langle W_{t_2}, W_{t_1}\right\rangle \in \mathsf{co}'$, which violates the acylicity of co'. We show that $\langle h, \mathsf{co}\rangle$ satisfies Prefix. Assume by contradiction that there exists a Prefix violation between t_1, t_2, t_3, t_4 (shown in Figure 10a), i.e., for some $x \in \mathsf{vars}(h)$, $\langle t_1, t_3\rangle \in \mathsf{wr}_x$ and t_2 writes x, $\langle t_1, t_2\rangle \in \mathsf{co}$, $\langle t_2, t_4\rangle \in \mathsf{co}^*$ and $\langle t_4, t_3\rangle \in \mathsf{wr} \cup \mathsf{so}$. Then, the corresponding transactions $W_{t_1}, W_{t_2}, W_{t_4}, R_{t_3}$ in $h_{R|W}$ would be related as follows: $\left\langle W_{t_1}, W_{t_2}\right\rangle \in \mathsf{co}'$ and $\left\langle W_{t_1}, R_{t_3}\right\rangle \in \mathsf{wr}_x{}'$ because $\langle t_1, t_3\rangle \in \mathsf{wr}_x$ and $\langle t_1, t_2\rangle \in \mathsf{co}$. Since co' satisfies Serializability, then $\left\langle R_{t_3}, W_{t_2}\right\rangle \in \mathsf{co}'$. But $\langle t_2, t_4\rangle \in \mathsf{co}^*$ and $\langle t_4, t_3\rangle \in \mathsf{wr} \cup \mathsf{so}$ imply that $\left\langle W_{t_2}, W_{t_4}\right\rangle \in \mathsf{co}'^*$ and $\left\langle W_{t_4}, R_{t_3}\right\rangle \in \mathsf{wr}' \cup \mathsf{so}'$, which show that co' is cyclic (the red cycle in Figure 10b), a contradiction. □

Since the history $h_{R|W}$ can be constructed in linear time, Lemma 4.4, Theorem 4.8, and Corollary 4.3 imply the following result.

Corollary 4.9. *For an arbitrary but fixed constant $k \in \mathbb{N}$, the problem of checking prefix consistency for histories of width at most k is polynomial time.*

4.3 Reducing Snapshot Isolation to Serializability

We extend the reduction of prefix consistency to serializability to the case of snapshot isolation. Compared to prefix consistency, snapshot isolation disallows transactions that read the same snapshot of the database to commit together if they write on a common variable (stated by the Conflict axiom). More precisely, for any pair of transactions t_1 and t_2 writing to a common variable, t_1 must observe the effects of t_2 or vice-versa. We refine the definition of $h_{R|W}$ such that any "serialization" (i.e.., commit order satisfying Serializability) disallows that the read transactions corresponding to two such transactions are ordered both before their write counterparts. We do this by introducing auxiliary variables that are read or written by these transactions. For instance, Figure 11 shows this transformation on the two histories in Figure 11a and Figure 11c, which represent the anomalies known as "lost update" and "write skew", respectively. The former is not admitted by SI while the latter is admitted. Concerning "lost update", the read counterpart of the transaction on the left writes to a variable x12 that is read by its write counterpart, but also written by the write counterpart of the other transaction. This forbids that the latter is serialized in between the read and write counterparts of the transaction on the left. A similar scenario is imposed on the transaction on the right, which makes that the transformed history is not serializable. Concerning



(a) Lost update (b) Lost update (transformed) (c) Write skew (d) Write skew (transformed)

Fig. 11. Reducing SI to SER.

the "write skew" anomaly, the transformed history is exactly as for the PC reduction since the two transactions don't write on a common variable. It is clearly serializable.

For a history $h = \langle T, \mathsf{wr}, \mathsf{so}\rangle$, the history $h^c_{R|W} = \langle T', \mathsf{wr}', \mathsf{so}'\rangle$ is defined as $h_{R|W}$ with the following additional construction: for every two transactions t_1 and $t_2 \in T$ that write on a common variable,

- R_{t_1} and W_{t_2} (resp., R_{t_2} and W_{t_1}) write on a variable $x_{1,2}$ (resp., $x_{2,1}$),
- the write transaction of t_i reads $x_{i,j}$ from the read transaction of t_i, for all $i \neq j \in \{1, 2\}$, i.e., $\mathsf{wr}_{x_{1,2}} = \{\langle R_{t_1}, W_{t_1}\rangle\}$ and $\mathsf{wr}_{x_{2,1}} = \{\langle R_{t_2}, W_{t_2}\rangle\}$.

Note that $h_{R|W}$ and $h^c_{R|W}$ have the same width (the session order is defined exactly in the same way), which implies, by Lemma 4.4, that h and $h^c_{R|W}$ have the same width.

The following result can be proved using similar reasoning as in the case of prefix consistency.

Theorem 4.10. *A history h satisfies snapshot isolation iff $h^c_{R|W}$ is serializable.*

Note that $h^c_{R|W}$ and h have the same width, and that $h^c_{R|W}$ can be constructed in linear time. Therefore, Theorem 4.10, and Corollary 4.3 imply the following result.

Corollary 4.11. *For an arbitrary but fixed constant $k \in \mathbb{N}$, the problem of checking snapshot isolation for histories of width at most k is polynomial time.*

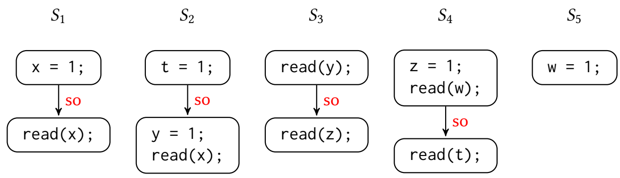
5 COMMUNICATION GRAPHS

In this section, we present an extension of the polynomial time results for PC, SI, and SER, which allows to handle histories where the sharing of variables between different sessions is *sparse*. For the results in this section, we take the simplifying assumption that the session order is a union of transaction sequences (modulo the fictitious transaction writing the initial values), i.e., each transaction sequence corresponding to the standard notion of *session* [9]. We represent the sharing of variables between different sessions using an undirected graph called a *communication graph*. For instance, the communication graph of the history in Figure 12a is given in Figure 12b. For readability, the edges are marked with the variables accessed by the two sessions.

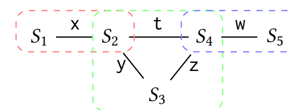
We show that the problem of checking PC, SI, or SER is polynomial time when the size of every *biconnected* component of the communication graph is bounded by a fixed constant. This is stronger than the results in Section 4 because the number of biconnected components can be arbitrarily large which means that the total number of sessions is unbounded. In general, we prove that the time complexity of these consistency criteria is exponential only in the maximum size of such a biconnected component, and not the whole number of sessions.

An undirected graph is biconnected if it is connected and if any one vertex were to be removed, the graph will remain connected, and a biconnected component of a graph G is a maximal biconnected

[9]The results can be extended to arbitrary session orders by considering maximal transaction sequences in session order instead of sessions.



(a) A history with 5 sessions.



(b) The communication graph and its decomposition in biconnected components.

Fig. 12. A history and its communication graph.

subgraph of G. Figure 12b shows the decomposition in biconnected components of a communication graph. This graph contains 5 sessions while every biconnected component is of size at most 3. Intuitively, any potential consistency violation associated to a history will contain a consistency violation that contained in sessions in the same biconnected component. Therefore, checking any of these criteria can be done in isolation for each biconnected component (more precisely, on sub-histories that contain only sessions in the same biconnected component). Actually, this decomposition argument works even for RC, RA, and CC. For instance, in the case of the history in Figure 12a, any consistency criterion can be checked looking in isolation at three sub-histories: a sub-history with S_1 and S_2, a sub-history with S_2, S_3, and S_4, and a sub-history with S_4 and S_5.

Formally, a *communication graph* of a history h is an undirected graph $\mathsf{Comm}(h) = (V, E)$ where the set of vertices V is the set of sessions in h [10], and $(v, v') \in E$ iff the sessions v and v' contain two transactions t_1 and t_2, respectively, such that t_1 and t_2 read or write a common variable x.

We begin with a technical lemma showing that *minimal* paths of certain form in the graph representing a history h and a relation co (on the transactions of h) lie within a single biconnected component of the underlying communication graph. This is used to show that any consistency violation can be exposed by looking at a single biconnected component at a time. The graph representing a history h and a relation co on the transactions of h is denoted by $\mathsf{G}(h, \mathsf{co})$ [11].

Given a graph $\mathsf{G}(h, \mathsf{co})$ and r a term over the relations so, wr, and co, e.g., $(\mathsf{wr} \cup \mathsf{so})^+$, a path of form r (or r-path) is a sequence of edges representing so, wr, or co dependencies as specified by the term r, e.g., a sequence of wr or so dependencies.

Lemma 5.1. *Let $B_1, \ldots, B_n$ be the biconnected components of $\mathsf{Comm}(h)$ for a history $h = \langle T, \mathsf{wr}, \mathsf{so}\rangle$. For each B_i, let co_i be a total order on the transactions of B_i [12] extending the session order so on the transactions of B_i. Also, let $\mathsf{co} = \bigcup_i \mathsf{co}_i$. Then, for every term $r \in \{\mathsf{co}^+, (\mathsf{wr} \cup \mathsf{so})^+\}$, any minimal r-path in the graph $\mathsf{G}(h, \mathsf{co})$ between two transactions from the same biconnected component includes only transactions of that biconnected component.*

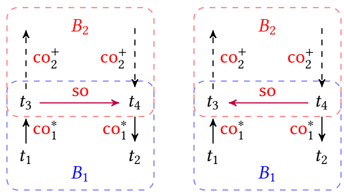
Proof. We consider the case $r = \mathsf{co}^+$. Consider a *minimal* co^+-path $\pi = t_0, \ldots, t_n$ between two transactions t_0 and t_n from the same biconnected component B of $\mathsf{Comm}(h)$ (i.e., from sessions in B). Assume by contradiction, that π traverses multiple biconnected components. We define a path $\pi_s = v_0, \ldots, v_m$ between sessions, i.e., vertices of $\mathsf{Comm}(h)$, which contains an edge (v_j, v_{j+1}) iff π contains an edge (t_i, t_{i+1}) with t_i a transaction of session v_j and t_{i+1} a transaction of session $v_{j+1} \neq v_j$. Since any graph decomposes to a forest of biconnected components, this path must necessarily leave and enter some biconnected component B_1 to and from the same biconnected component B_2, i.e., π_s must contain two vertices v_{j_1} and v_{j_2} in B_1 such that the successor v_{j_1+1} of v_{j_1} and the predecessor v_{j_2-1} of v_{j_2} are from B_2. Let t_1, t_2, t_3, t_4 be the transactions in the

[10]The transaction writing the initial values is considered as a distinguished session.
[11]The nodes of $\mathsf{G}(h, \mathsf{co})$ correspond to transactions in h and the edges connect pairs of transactions in so, wr, or co.
[12]That is, transactions that are included in the sessions in B_i.

path π corresponding to v_{j_1}, v_{j_2}, v_{j_1+1}, and v_{j_2-1}, respectively. Now, since any two biconnected components share at most one vertex, it follows that t_3 and t_4 are from the same session and

- if $\langle t_3, t_4 \rangle \in \mathsf{so}$, then there exists a shorter path between t_0 and t_1 that uses the so relation between $\langle t_3, t_4 \rangle$ (we recall that $\mathsf{so} \subseteq \bigcup_i \mathsf{co}_i$) instead of the transactions in B_2, pictured in Figure 13a, which is a contradiction to the minimality of π,
- if $\langle t_4, t_3 \rangle \in \mathsf{so}$, then, we have a cycle in $\bigcup_i \mathsf{co}_i \cup \mathsf{so}$, pictured in Figure 13b, which is also a contradiction.

The case $r = (\mathsf{wr} \cup \mathsf{so})^+$ can be proved in a similar manner since the reasoning outlined in Figure 13 reduces to short-circuiting a path using a single so edge (and so is included in $(\mathsf{wr} \cup \mathsf{so})^+$). ◻



(a) $\langle t_3, t_4 \rangle \in \mathsf{so}$ (b) $\langle t_4, t_3 \rangle \in \mathsf{so}$

Fig. 13. Minimal paths between transactions in the same biconnected component.

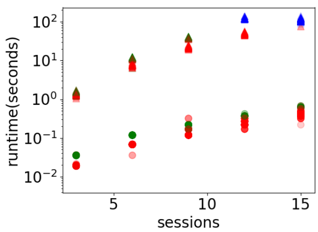
Now we prove our final claim. For a history $h = (T, \mathsf{so}, \mathsf{wr})$ and biconnected component B of $\mathsf{Comm}(h)$, the projection of h over transactions in sessions of B is denoted by $h \downarrow B$, i.e., $h \downarrow B = (T', \mathsf{so}', \mathsf{wr}')$ where T' is the set of transactions in sessions of B, so' and wr' are the projections of so and wr, respectively, on T'.

Theorem 5.2. *For any criterion $C \in \{RA, RC, CC, PC, SI, SER\}$, a history h satisfies C iff for every biconnected component B of* $\mathsf{Comm}(h)$*, $h \downarrow B$ satisfies C.*

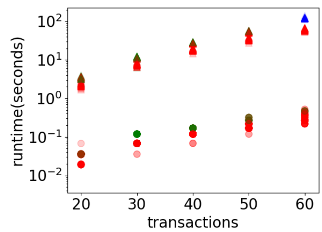
Proof. The "only-if" direction is obvious. For the "if" direction, we first consider the cases $C \in \{\mathrm{RA}, \mathrm{RC}, \mathrm{CC}, \mathrm{SER}\}$. The proof concerning PC and SI is based on the reduction to SER outlined in Section 4.2 and Section 4.3, respectively, and it is given afterwards. Let $B_1, \ldots, B_n$ be the biconnected components of $\mathsf{Comm}(h)$.

Let $C \in \{\mathrm{RA}, \mathrm{RC}, \mathrm{CC}, \mathrm{SER}\}$ and let co_i be the commit order that witnesses that $h \downarrow B_i$ satisfies C, for each $1 \leq i \leq n$. The union $\bigcup_i \mathsf{co}_i$ is acyclic since otherwise, any minimal cycle would be a minimal path between transactions of the same biconnected component B_j, and, by Lemma 5.1, it will include only transactions of B_j which is a contradiction to co_j being a total order. We show that any linearization co of $\bigcup_i \mathsf{co}_i$ along with h satisfies the axioms of C. The axioms defining RA, RC, CC, and SER involve transactions that write or read a common variable, which implies that they belong to the same biconnected component (we refer to the transactions t_1, t_2, and t_3 in Figure 2). Furthermore, by Lemma 5.1, minimal paths witnessing the dependencies in those axioms, e.g., $(\mathsf{wr} \cup \mathsf{so})^+$ for CC, are also formed of transactions included in the same biconnected component. Therefore, co satisfies any of those axioms provided that each co_i does.

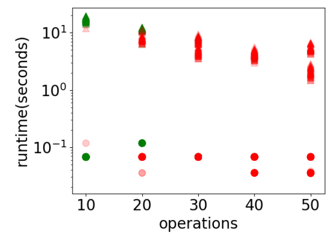
We now consider the case where $C = \mathrm{PC}$. Assume that each B_i satisfies PC. Based on the reduction in Section 4.2, h satisfies PC iff $h_{R|W}$ satisfies SER. Moreover, since $h_{R|W}$ is obtained from h by splitting each transaction t into a read transaction R_t and a write transaction W_t while keeping all session order dependencies, each session in h corresponds to a session in $h_{R|W}$ that reads or writes exactly the same set of variables. Therefore, $\mathsf{Comm}(h)$ is isomorphic to $\mathsf{Comm}(h_{R|W})$. Since B_i satisfies PC, we get that the corresponding biconnected component B'_i of $\mathsf{Comm}(h_{R|W})$ satisfies SER, for every i. Therefore, $h_{R|W}$ satisfies SER, which implies that h satisfies PC. The case of SI is proved in a similar way using the reduction to the serializability of $h^c_{R|W}$ presented in Section 4.3 (note that two transactions of $h^c_{R|W}$ may read or write an additional common variable only if they were writing a common variable in the original history and therefore, $\mathsf{Comm}(h)$ is still isomorphic to $\mathsf{Comm}(h^c_{R|W})$). ◻



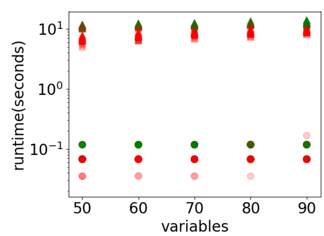
(a) Sessions.



(b) Transactions per session.



(c) Operations per transaction.



(d) Variables.

Fig. 14. Scalability of our Serializability checking algorithm (Algorithm 2), and a comparison to a SAT encoding. The x-axis represents the varying parameter while the y-axis represents the wall clock time in logarithmic scale. The circular, resp., triangle, dots represent wall clock times of our algorithm, resp., the SAT encoding. The red, green, and blue dots represent invalid, valid and resource exhausted instances, respectively.

Since the decomposition of a graph into biconnected components can be done in linear time, Theorem 5.2 implies that any of the criteria PC, SI, or SER can be checked in time $O(\text{size}(h)^{\text{bi-size}(h)} \cdot \text{size}(h)^3 \cdot \text{bi-nb}(h))$ where $\text{bi-size}(h)$ and $\text{bi-nb}(h)$ are the maximum size of a biconnected component in $\text{Comm}(h)$ and the number of biconnected components of $\text{Comm}(h)$, respectively. The following corollary is a direct consequence of this observation.

Corollary 5.3. *For an arbitrary but fixed constant $k \in \mathbb{N}$ and any criterion $C \in \{PC, SI, SER\}$, the problem of checking if a history h satisfies C is polynomial time, provided that the size of every biconnected component of* $\text{Comm}(h)$ *is bounded by k.*

6 EXPERIMENTAL EVALUATION

To demonstrate the practical value of the theory developed in the previous sections, we argue that our algorithms:

- are efficient and scalable,
- enable an effective testing framework allowing to expose consistency violations in production databases.

We focus on three of the criteria introduced in Section 2: *serializability* which is NP-complete in general and polynomial time when the number of sessions is considered to be a constant, *snapshot isolation* which can be reduced in linear time to serializability, and *causal consistency* which is polynomial time in general. As benchmark, we consider histories extracted from three distributed databases: CockroachDB [3], Galera [5], and AntidoteDB [8]. Following the approach in Jepsen [1], histories are generated with random clients. For the experiments described hereafter, the randomization process is parametrized by: (1) the number of sessions (**#sess**), (2) the number of transactions per session (**#trs**), (3) the number of operations per transaction (**#ops**), and (4) an upper bound on the number of used variables (**#vars**) [13]. For any valuation of these parameters, half of the histories generated with CockroachDB and Galera are restricted such that the sets of variables written by any two sessions are disjoint (the sets of read variables are not constrained). This restriction is used to increase the frequency of valid histories.

In a first experiment, we investigated the efficiency of our serializability checking algorithm (Algorithm 2) and we compared its performance with a direct SAT encoding [14] of the serializability definition in Section 2 (we used MiniSAT [18] to solve the SAT queries). We used histories extracted from CockroachDB which claims to implement serializability, acknowledging however the possibility of anomalies [4]. The sessions of a history are uniformly distributed among 3 nodes of a single cluster. To evaluate scalability, we fix a reference set of parameter values: **#sess**=6, **#trs**=30, **#ops**=20, and **#vars** = 60 × **#sess**, and vary only one parameter at a time. For instance, the number of sessions varies from 3 to 15 in increments of 3. We consider 100 histories for each combination of parameter values. The experimental data is reported in Figure 14. Our algorithm scales well even when increasing the number of sessions, which is not guaranteed by its worst-case complexity (in general, this is exponential in the number of sessions). Also, our algorithm is at least two orders of magnitude more efficient than the SAT encoding. While the performance of SAT solvers is known to be heavily affected by the specific encoding of the problem, we strove to make the SAT formula as succinct as possible and optimize its construction. We have fixed a 10 minutes timeout, a limit of 10GB of memory, and a limit of 10GB on the files containing the formulas to be passed to the SAT solver. The blue dots represent resource exhausted instances. The SAT encoding reaches the file limit for 148 out of 200 histories with at least 12 sessions (Figure 14a) and for 50 out of 100 histories with 60 transactions per session (Figure 14b), the other parameters being fixed as explained above.

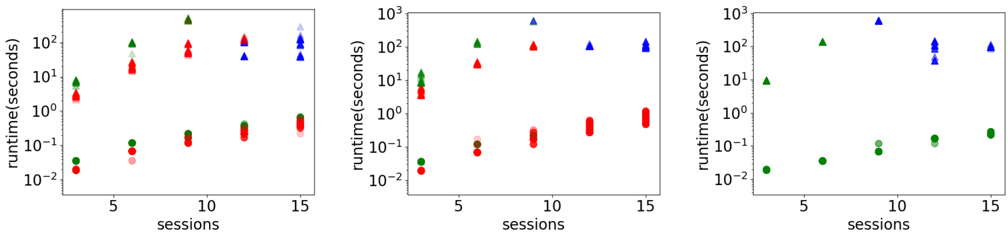
We have found a large number of violations, whose frequency increases with the number of sessions, transactions per session, or operations per transaction, and decreases when allowing more variables. This is expected since increasing any of the former parameters increases the chance of interference between different transactions while increasing the latter has the opposite effect. The second and third column of Table 2 give a more precise account of the kind of violations we found by identifying for each criterion X, the number of histories that violate X but no other criterion weaker than X, e.g., there is only one violation to SI that satisfies PC.

The second experiment measures the scalability of the SI checking algorithm obtained by applying the reduction to SER described in Section 4.3 followed by the SER checking algorithm in Algorithm 2, and its performance compared to a SAT encoding of SI. Actually, the reduction to SER is performed on-the-fly, while traversing the history and checking for serializability (of the transformed history). The SAT encoding follows the same principles as in the case of serializability. We focus on its behavior when increasing the number of sessions (varying the other parameters leads to similar results). As benchmark, we used the same CockroachDB histories as in Figure 14a and a number of histories extracted from Galera[15] whose documentation contains contradicting claims about whether it implements snapshot isolation [6, 7]. We use 100 histories per combination of parameter values as in the previous experiment. The results are reported in Figure 15a and Figure 15b. We observe the same behavior as in the case of SER. In particular, the SAT encoding reaches the file limit for 150 out of 200 histories with at least 12 sessions in the case of the CockroachDB histories,

[13]We ensure that every value is written at most once.

[14]For each ordered pair of transactions t_1, t_2 we add two propositional variables representing $\langle t_1, t_2\rangle \in (\mathsf{wr} \cup \mathsf{so})^+$ and $\langle t_1, t_2\rangle \in \mathsf{co}$, respectively. Then we generate clauses corresponding to: (1) singleton clauses defining the relation $\mathsf{wr} \cup \mathsf{so}$ (extracted from the input history), (2) $\langle t_1, t_2\rangle \in \mathsf{wr} \cup \mathsf{so}$ implies $\langle t_1, t_2\rangle \in \mathsf{co}$, (3) co being a total order, and (4) the axioms corresponding to the considered consistency model. This is an optimization that does not encode wr and so separately, which is sound because of the shape of our axioms (and because these relations are fixed apriori).

[15]In order to increase the frequency of valid histories, all sessions are executed on a single node.



(a) Checking SI (CockroachDB) (b) Checking SI (Galera) (c) Checking CC (AntidoteDB)

Fig. 15. Scalability of our SI checking algorithm (Section 4.3) and CC checking algorithm (Algorithm 1), and a comparison to a SAT encoding. The x-axis represents the varying parameter while the y-axis represents the wall clock time in logarithmic scale. The circular, resp., triangle, dots represent wall clock times of our algorithm, resp., the SAT encoding. The red, green, and blue dots represent invalid, valid and resource exhausted instances, respectively.

and for 162 out of 300 histories with at least 9 sessions in the case of the Galera histories. The last two columns in Table 2 classify the set of violations depending on the weakest criterion that they violate.

We also evaluated the performance of the CC checking algorithm in Section 3 when increasing the number of sessions, on histories extracted from AntidoteDB, which claims to implement causal consistency [9]. The results are reported in Figure 15c. In this case, the SAT encoding reaches the file limit for 150 out of 300 histories with at least 9 sessions. All the histories considered in this experiment are valid. However, when experimenting with other parameter values, we have found several violations. The smallest parameter values for which we found violations were 3 sessions, 14 transactions per session, 14 operations per transaction, and 5 variables. The violations we found are also violations of Read Atomic. For instance, one of the violations contains two transactions t_1 and t_2, each of them writing to two variables x_1 and x_2, and another transaction t_3 that reads x_1 from t_1 and x_2 from t_2 (t_1 and t_2 are from different sessions while t_3 is an so successor of t_1 in the same session). These violations are novel and they were confirmed by the developers of AntidoteDB.

The refinement of the algorithms above based on communication graphs, described in Section 5, did not have a significant impact on their performance. The histories we generated contained few biconnected components (many histories contained just a single biconnected component) which we believe is due to our proof of concept deployment of these databases on a single machine that did not allow to experiment with very large number of sessions and variables.

7 RELATED WORK

Cerone et al. [16] give the first formalization of the criteria we consider in this paper, using the specification methodology of Burckhardt et al. [14]. This formalization uses two auxiliary relations, a *visibility* relation which represents the fact that a transaction "observes" the effects of another transaction and a *commit order*, also called arbitration order, like in our case. Executions are abstracted using a notion of history that includes only a session order and the adherence to some consistency criterion is defined as the existence of a visibility relation and a commit order satisfying certain axioms. Motivated by practical goals, our histories include a write-read relation, which enables more uniform and in our opinion, more intuitive, axioms to characterize consistency criteria. Our formalizations are however equivalent with those of Cerone et al. [16]. Moreover, Cerone et al. [16] do not investigate algorithmic issues as in our paper.

	Serializability checking		Snapshot Isolation checking	
Weakest criterion violated	CockroachDB (disjoint writes)	CockroachDB (no constraints)	Galera (disjoint writes)	Galera (no constraints)
Read Committed			19	50
Read Atomic	180	547	91	139
Causal Consistency	339	382	88	43
Prefix Consistency	2	7		
Snapshot Isolation		1		1
Serializability	25			
Total number of violations	546/1000	937/1000	198/250	233/250

Table 2. Violation statistics. The "disjoint writes" columns refer to histories where the set of variables written by any two sessions are disjoint.

Papadimitriou [25] showed that checking serializability of an execution is NP-complete. Moreover, it identifies a stronger criterion called *conflict serializability* which is polynomial-time checkable. Conflict serializability assumes that histories are given as sequences of operations and requires that the commit order be consistent with a *conflict-order* between transactions defined based on this sequence (roughly, a transaction t_1 is before a transaction t_2 in the conflict order if it accesses some variable x before t_2 does). This result is not applicable to distributed databases where deriving such a sequence between operations submitted to different nodes in a network is impossible.

Bouajjani et al. [13] showed that checking several variations of causal consistency on executions of a *non-transactional* distributed database is polynomial time (they also assume that every value is written at most once). Assuming singleton transactions, our notion of CC corresponds to the causal convergence criterion in Bouajjani et al. [13]. Therefore, our result concerning CC can be seen as an extension of this result concerning causal convergence to transactions.

There are some works that investigated the problem of checking consistency criteria like sequential consistency and linearizability in the case of shared-memory systems. Gibbons and Korach [21] showed that checking linearizability of the single-value register type is NP-complete in general, but polynomial time for executions where every value is written at most once. Using a reduction from serializabilty, they showed that checking sequential consistency is NP-complete even when every value is written at most once. Emmi and Enea [19] extended the result concerning linearizability to a series of abstract data types called collections, that includes stacks, queues, key-value maps, etc. Sequential consistency reduces to serializability for histories with singleton transactions (i.e., formed of a single read or write operation). Therefore, our polynomial time result for checking serializability of bounded-width histories (Corollary 4.3) implies that checking sequential consistency of histories with a bounded number of threads is polynomial time. The latter result has been established independently by Abdulla et al. [10].

The notion of *communication graph* is inspired by the work of Chalupa et al. [17] which investigates partial-order reduction (POR) techniques for multi-threaded programs. In general, the goal of partial-order reduction [20] is to avoid exploring executions which are equivalent w.r.t. some suitable notion of equivalence, e.g., Mazurkiewicz trace equivalence [23]. They use the acyclicity of communication graphs to define a class of programs for which their POR technique is optimal. The algorithmic issues they explore are different than ours and they don't investigate biconnected components of this graph as in our results.

8 CONCLUSIONS

Our results provide an effective means of checking the correctness of transactional databases with respect to a wide range of consistency criteria, in an efficient way. We devise a new specification framework for these criteria, which besides enabling efficient verification algorithms, provide a novel understanding of the differences between them in terms of set of transactions that *must* be committed before a transaction which is read during the execution. These algorithms are shown to be scalable and orders of magnitude more efficient than standard SAT encodings of these criteria (as defined in our framework). While the algorithms are quite simple to understand and implement, the proof of their correctness is non-trivial and benefits heavily from the new specification framework. One important venue for future work is identifying root causes for a given violation. The fact that we are able to deal with a wide range of criteria is already helpful in identifying the weakest criterion that is violated in a given execution. Then, in the case of RC, RA, and CC, where inconsistencies correspond to cycles in the commit order, the root cause could be attributed to a minimal cycle in this relation. We did this in our communication with the Antidote developers to simplify the violation we found which contained 42 transactions. In the case of PC, SI, and SER, it could be possible to implement a search procedure similar to CDCL in SAT solvers, in order to compute the root-cause as a SAT solver would compute an unsatisfiability core.

ACKNOWLEDGMENTS

This work is supported in part by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 678177).

REFERENCES

[1] [n. d.]. http://jepsen.io Retrieved March 28th, 2019.

[2] [n. d.]. https://github.com/jepsen-io/jepsen/blob/master/galera/src/jepsen/galera/dirty_reads.clj Retrieved March 28th, 2019.

[3] [n. d.]. https://github.com/cockroachdb/cockroach Retrieved March 28th, 2019.

[4] [n. d.]. https://www.cockroachlabs.com/docs/v2.1/transactions.html#isolation-levels Retrieved March 28th, 2019.

[5] [n. d.]. http://galeracluster.com Retrieved March 28th, 2019.

[6] [n. d.]. http://galeracluster.com/documentation-webpages/faq.html Retrieved March 28th, 2019.

[7] [n. d.]. http://galeracluster.com/documentation-webpages/isolationlevels.html#intra-node-vs-inter-node-isolation-in-galera-cluster Retrieved March 28th, 2019.

[8] [n. d.]. https://www.antidotedb.eu Retrieved March 28th, 2019.

[9] [n. d.]. https://antidotedb.gitbook.io/documentation/overview/configuration Retrieved March 28th, 2019.

[10] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Bengt Jonsson, Magnus Lång, Tuan Phong Ngo, and Konstantinos (Kostis) Sagonas. 2019. Optimal Stateless Model Checking for Read-from Equivalence under Sequential Consistency. *PACMPL* OOPSLA.

[11] Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, and Patrick E. O'Neil. 1995. A Critique of ANSI SQL Isolation Levels. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA, May 22-25, 1995.*, Michael J. Carey and Donovan A. Schneider (Eds.). ACM Press, 1–10. https://doi.org/10.1145/223784.223785

[12] Ranadeep Biswas and Constantin Enea. 2019. On the Complexity of Checking Transactional Consistency. abs/1908.04509 (2019). arXiv:1908.04509 https://arxiv.org/abs/1908.04509

[13] Ahmed Bouajjani, Constantin Enea, Rachid Guerraoui, and Jad Hamza. 2017. On verifying causal consistency. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 626–638. http://dl.acm.org/citation.cfm?id=3009888

[14] Sebastian Burckhardt, Alexey Gotsman, Hongseok Yang, and Marek Zawirski. 2014. Replicated data types: specification, verification, optimality. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, Suresh Jagannathan and Peter Sewell (Eds.). ACM, 271–284. https://doi.org/10.1145/2535838.2535848

[15] Sebastian Burckhardt, Daan Leijen, Jonathan Protzenko, and Manuel Fähndrich. 2015. Global Sequence Protocol: A Robust Abstraction for Replicated Shared State. In *29th European Conference on Object-Oriented Programming, ECOOP 2015, July 5-10, 2015, Prague, Czech Republic (LIPIcs)*, John Tang Boyland (Ed.), Vol. 37. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 568–590. https://doi.org/10.4230/LIPIcs.ECOOP.2015.568

[16] Andrea Cerone, Giovanni Bernardi, and Alexey Gotsman. 2015. A Framework for Transactional Consistency Models with Atomic Visibility. In *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015 (LIPIcs)*, Luca Aceto and David de Frutos-Escrig (Eds.), Vol. 42. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 58–71. https://doi.org/10.4230/LIPIcs.CONCUR.2015.58

[17] Marek Chalupa, Krishnendu Chatterjee, Andreas Pavlogiannis, Nishant Sinha, and Kapil Vaidya. 2018. Data-centric dynamic partial order reduction. *PACMPL* 2, POPL (2018), 31:1–31:30. https://doi.org/10.1145/3158119

[18] Niklas Eén and Niklas Sörensson. 2003. An Extensible SAT-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers (Lecture Notes in Computer Science)*, Enrico Giunchiglia and Armando Tacchella (Eds.), Vol. 2919. Springer, 502–518. https://doi.org/10.1007/978-3-540-24605-3_37

[19] Michael Emmi and Constantin Enea. 2018. Sound, complete, and tractable linearizability monitoring for concurrent collections. *PACMPL* 2, POPL (2018), 25:1–25:27. https://doi.org/10.1145/3158113

[20] Cormac Flanagan and Patrice Godefroid. 2005. Dynamic partial-order reduction for model checking software. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005, Long Beach, California, USA, January 12-14, 2005*, Jens Palsberg and Martín Abadi (Eds.). ACM, 110–121. https://doi.org/10.1145/1040305.1040315

[21] Phillip B. Gibbons and Ephraim Korach. 1997. Testing Shared Memories. *SIAM J. Comput.* 26, 4 (1997), 1208–1244. https://doi.org/10.1137/S0097539794279614

[22] Leslie Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* 21, 7 (1978), 558–565. https://doi.org/10.1145/359545.359563

[23] Antoni W. Mazurkiewicz. 1986. Trace Theory. In *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, Germany, 8-19 September 1986 (Lecture Notes in Computer Science)*, Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg (Eds.), Vol. 255. Springer, 279–324. https://doi.org/10.1007/3-540-17906-2_30

[24] Burcu Kulahcioglu Ozkan, Rupak Majumdar, Filip Niksic, Mitra Tabaei Befrouei, and Georg Weissenbacher. 2018. Randomized testing of distributed systems with probabilistic guarantees. *PACMPL* 2, OOPSLA (2018), 160:1–160:28. https://doi.org/10.1145/3276530

[25] Christos H. Papadimitriou. 1979. The serializability of concurrent database updates. *J. ACM* 26, 4 (1979), 631–653. https://doi.org/10.1145/322154.322158

[26] Douglas B. Terry, Alan J. Demers, Karin Petersen, Mike Spreitzer, Marvin Theimer, and Brent B. Welch. 1994. 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*. IEEE Computer Society, 140–149. https://doi.org/10.1109/PDIS.1994.331722

[27] Pierre Wolper. 1986. Expressing Interesting Properties of Programs in Propositional Temporal Logic. In *Conference Record of the Thirteenth Annual ACM Symposium on Principles of Programming Languages, St. Petersburg Beach, Florida, USA, January 1986*. ACM Press, 184–193. https://doi.org/10.1145/512644.512661