A CORRECTNESS PROOFS OF THE VERIFICATION ALGORITHMS

Lemma 3. *If the graph $\mathcal{G}$ returned by* BuildDependency *does not contain any* WW *loops or any instances of the* Divergence *pattern, then it is a dependency graph of $\mathcal{H}$.*

Proof. Suppose that $\mathcal{G}$ does not contain any WW loops or any instances of the Divergence pattern. We need to show that for each object $k \in \mathsf{X}$, $\mathrm{WW}(k)$ is a strict total order on the set $\mathsf{WriteTx}_x$. Suppose by contradiction that there exists an object $x \in \mathsf{X}$ such that $\mathrm{WW}(x)$ is not a strict total order on $\mathsf{WriteTx}_x$. In the following, we identify an instance of the Divergence pattern in $\mathcal{G}$.

Since $\mathcal{G}$ does not contain any WW loops, $\mathrm{WW}(x)$ is a strict order. Therefore, $\mathrm{WW}(x)$ is not total. Consider two transactions T_a and T_b such that they both write to x with different values, but there are no $\mathrm{WW}(x)$ dependency edges between them in $\mathcal{G}$; see Figure 15. If T_a and T_b read the value of x from the same transaction, then we are done. Otherwise, consider the transactions $T_{a'}$ and $T_{b'}$ ($T_{a'} \neq T_{b'}$) from which T_a and T_b read the value of x, respectively. That is, $T_{a'} \xrightarrow{\mathrm{WR}(x)/\mathrm{WW}(x)} T_a$ and $T_{b'} \xrightarrow{\mathrm{WR}(x)/\mathrm{WW}(x)} T_b$ (see line 11 of BuildDependency). It is clear that $T_{a'} \neq T_b$ and $T_{b'} \neq T_a$, since otherwise there would be a $\mathrm{WW}(x)$ dependency edge between T_a and T_b. There are two cases regarding whether there is a $\mathrm{WW}(x)$ dependency edge between $T_{a'}$ and $T_{b'}$ in $\mathcal{G}$:

- Suppose there is a $\mathrm{WW}(x)$ dependency edge between $T_{a'}$ and $T_{b'}$ in $\mathcal{G}$, say $T_{a'} \xrightarrow{\mathrm{WW}(x)} T_{b'}$. Note that this $\mathrm{WW}(x)$ edge may be derived by transitivity (see line 13 of BuildDependency). Let $T_{c'}$ be a transaction *directly following* $T_{a'}$ in the $\mathrm{WW}(x)$ dependency chain from $T_{a'}$ to $T_{b'}$. That is, the edge $T_{a'} \xrightarrow{\mathrm{WW}(x)} T_{c'}$ coincides with $T_{a'} \xrightarrow{\mathrm{WR}(x)} T_{b'}$ (see line 11 of BuildDependency). As illustrated in Figure 15, we have $T_{a'} \xrightarrow{\mathrm{WR}(x)/\mathrm{WW}(x)} T_{c'} \xrightarrow{\mathrm{WW}(x)} T_{b'}$. It is possible that $T_{c'} = T_{b'}$, but we argue that
  - $T_{c'} \neq T_a$. Otherwise, we have $T_a = T_{c'} \xrightarrow{\mathrm{WW}(x)} T_{b'} \xrightarrow{\mathrm{WW}(x)} T_b$. Then by transitivity, $T_a \xrightarrow{\mathrm{WW}(x)} T_b$, contradicting the assumption that there is no $\mathrm{WW}(x)$ dependency edge between T_a and T_b.
  - $T_{c'} \neq T_b$. Otherwise, we have $T_b = T_{c'} \xrightarrow{\mathrm{WW}(x)} T_{b'} \xrightarrow{\mathrm{WW}(x)} T_b$. Then by transitivity, $T_b \xrightarrow{\mathrm{WW}(x)} T_b$, contradicting the assumption that $\mathcal{G}$ contains no $\mathrm{WW}(x)$ loops.

  Therefore, $T_{a'}$, $T_{c'}$, and T_a forms an instance of the Divergence pattern.
- If there is no a $\mathrm{WW}(x)$ dependency edge between $T_{a'}$ and $T_{b'}$ in $\mathcal{G}$, we apply the same argument above to $T_{a'}$ and $T_{b'}$.

Since the number of transactions in $\mathcal{H}$ is finite and $\mathcal{H}$ contains an initial transaction $\perp_T$ which install initial values to all objects, we will eventually identify three transactions that form an instance of the Divergence pattern. □

Lemma 4. *If the graph $\mathcal{G}$ returned by* BuildDependency *is acyclic, then it is a dependency graph of $\mathcal{H}$.*

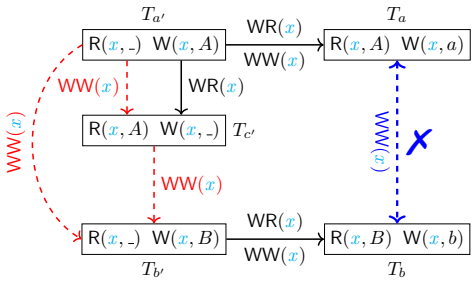


Figure 15: Transactions T_a and $T_{c'}$ read the same value of x from $T_{a'}$ and then write different values.

Proof. Suppose by contradiction that $\mathcal{G}$ is not a dependency graph of $\mathcal{H}$. By Lemma 3, $\mathcal{G}$ contains WW loops or instances of the Divergence pattern. Since $\mathcal{G}$ is acyclic, there are no WW loops in $\mathcal{G}$. Thus, $\mathcal{G}$ must contain instances of the Divergence pattern, as illustrated in Figure 6. However, in this case, there would be a cycle $T_2 \xrightarrow{\mathrm{RW}(x)} T_3 \xrightarrow{\mathrm{RW}(x)} T_2$ in $\mathcal{G}$. □

Theorem 6 (Correctness of CheckSSER). CheckSSER *returns* true *if and only if $\mathcal{H}$ satisfies* SSER.

Proof. The proof proceeds in two directions.

- (“ $\Longrightarrow$ ”) Suppose that CheckSSER returns true. Then the graph $\mathcal{G}$ at line 2 of CheckSSER is acyclic. By Lemma 4, $\mathcal{G}$ is a dependency graph of $\mathcal{H}$. By Definition 7, $\mathcal{H}$ satisfies SSER.
- (“ $\Longleftarrow$ ”) Suppose that CheckSSER returns false. Then the graph $\mathcal{G}$ at line 2 of CheckSSER is cyclic. Since $\mathcal{G}$ is a subgraph of any dependency graph of $\mathcal{H}$, no dependency graph of $\mathcal{H}$ is acyclic. By Definition 7, $\mathcal{H}$ does not satisfy SSER.

□

Similarly, by Lemma 4 and Definition 8, we can establish the correctness of CheckSER. The proof is omitted here.

Theorem 7 (Correctness of CheckSER). CheckSER *returns* true *if and only if $\mathcal{H}$ satisfies* SER.

Finally, we show the correctness of CheckSI.

Theorem 8 (Correctness of CheckSI). CheckSI *returns* true *if and only if $\mathcal{H}$ satisfies* SI.

Proof. The proof proceeds in two directions.

- (“ $\Longleftarrow$ ”) We show that if CheckSI returns false, then $\mathcal{H}$ does not satisfy SI. There are two cases:
  - Suppose that CheckSI returns false at line 3 of CheckSI. Then $\mathcal{G}$ contains an instance of the Divergence pattern. By Lemma 1, $\mathcal{H}$ does not satisfy SI.
  - Suppose that CheckSI returns false at line 6 of CheckSI. There are two cases regarding whether $\mathcal{G}$ contains a WW loop:
    - If $\mathcal{G}'$ contains a WW loop, then, by the construction of $\mathcal{G}'$ (line 5 of CheckSI), the graph $\mathcal{G}$ at line 5 also contains a WW loop. Since $\mathcal{G}$ is a

subgraph of any dependency graph of $\mathcal{H}$, $\mathcal{H}$ does not admit any legal dependency graphs. By Definition 9, $\mathcal{H}$ does not satisfy SI.

  - If $\mathcal{G}'$ contains no WW loops, then, by the construction of $\mathcal{G}'$ (line 5 of CheckSI), $\mathcal{G}$ does not contain any WW loop either. On the other hand, the graph $\mathcal{G}$ at line 5 does not contain any instances of the Divergence pattern. By Lemma 3, the graph $\mathcal{G}$ at line 5 is a dependency graph of $\mathcal{H}$. Since $\mathcal{G}$ is a subgraph of any dependency graph of $\mathcal{H}$, $\mathcal{G}$ is the unique dependency graph of $\mathcal{H}$. Then by Definition 9, $\mathcal{H}$ does not satisfy SI.

- (" $\Longrightarrow$ ") If CheckSI returns true at line 6 of CheckSI, then $\mathcal{G}'$ at line 5 of CheckSI is acyclic. Therefore, $\mathcal{G}'$ contains no WW loops. By the line of reasoning above, $\mathcal{G}$ at line 5 is a dependency graph of $\mathcal{H}$. Therefore, by Definition 9, $\mathcal{H}$ satisfies SI.

□

B CORRECTNESS PROOFS OF THE OPTIMIZED VERIFICATION ALGORITHMS

Proof of Lemma 2. For the $S \xrightarrow{\widehat{\mathrm{RW}}} T'$ edge in $\mathcal{G}$, there exists a transaction T such that $T \xrightarrow{\mathrm{WR}} S$ and $T \xrightarrow{\widehat{\mathrm{WW}}} T'$. Therefore, there exists a sequence of WW edges from T to T', denoted $T \xrightarrow{\mathrm{WW}} T_1 \xrightarrow{\mathrm{WW}} T_2 \xrightarrow{\mathrm{WW}} \cdots \xrightarrow{\mathrm{WW}} T'$ such that $S \xrightarrow{\mathrm{RW}} T_1$. □

The correctness of the optimized versions of CheckSSER and CheckSER directly follows from the following theorem.

Proof of Theorem 4. The proof proceeds in two directions.

- " $\Longrightarrow$ ": Suppose that $\mathcal{G}$ is acyclic. Since $E_{\widehat{\mathcal{G}}} \subseteq E_{\mathcal{G}}$, $\widehat{\mathcal{G}}$ is also acyclic.
- " $\Longleftarrow$ ": Suppose by contradiction that $\mathcal{G}$ contains a cycle, denoted C. We need to show that $\widehat{\mathcal{G}}$ also contains a cycle. If $E_C \subseteq E_{\widehat{\mathcal{G}}'}$, then we are done. Otherwise, any edge in $E_C \setminus E_{\widehat{\mathcal{G}}}$ must be either a $\widehat{\mathrm{WW}}$ edge or a $\widehat{\mathrm{RW}}$ edge. We can transform C into a cycle within $\widehat{\mathcal{G}}$ as follows (see Figure 7):
  - For each $\widehat{\mathrm{WW}}$ edge in C, replace it with a sequence of WW edges from which the $\widehat{\mathrm{WW}}$ edge can be derived by transitivity.
  - For each $S \xrightarrow{\widehat{\mathrm{RW}}} T'$ edge in C, replace it with a path $S \xrightarrow{\mathrm{RW}} T_1 \xrightarrow{\mathrm{WW}} T_2 \xrightarrow{\mathrm{WW}} \cdots \xrightarrow{\mathrm{WW}} T'$, as indicated by Lemma 2.

□

Let $\widehat{\mathcal{G}}'$ be the induced graph of $\widehat{\mathcal{G}}$, i.e., $\widehat{\mathcal{G}}' \leftarrow (V_{\widehat{\mathcal{G}}}, (\mathrm{SO}_{\widehat{\mathcal{G}}} \cup \mathrm{WR}_{\widehat{\mathcal{G}}} \cup \mathrm{WW}_{\widehat{\mathcal{G}}})\ ;\ \mathrm{RW}_{\widehat{\mathcal{G}}}?)$. The correctness of the optimized versions of CheckSI directly follows from the following theorem.

Proof of Theorem 5. The proof proceeds in two directions.

- " $\Longrightarrow$ ": Suppose that $\mathcal{G}'$ is acyclic. Since $E_{\widehat{\mathcal{G}}'} \subseteq E_{\mathcal{G}'}$, $\widehat{\mathcal{G}}'$ is also acyclic.
- " $\Longleftarrow$ ": Suppose by contradiction that $\mathcal{G}'$ contains a cycle, denoted C'. We need to show that $\widehat{\mathcal{G}}'$ also contains a cycle. If $E_{C'} \subseteq E_{\widehat{\mathcal{G}}'}$, then we are done. Otherwise, we can transform C' into a cycle within $\widehat{\mathcal{G}}'$ by replacing each edge in $E_{C'} \setminus E_{\widehat{\mathcal{G}}'}$ with edge(s) in $\widehat{\mathcal{G}}'$ as follows:
  (1) Consider a $\widehat{\mathrm{WW}}$ edge in C'. Replace it with a sequence of WW edges from which the $\widehat{\mathrm{WW}}$ edge can be derived by transitivity.
  (2) Consider an $S' \xrightarrow{\mathrm{SO}\ ;\ \widehat{\mathrm{RW}}} T'$ edge in C'. Suppose that it is derived from $S' \xrightarrow{\mathrm{SO}} S \xrightarrow{\widehat{\mathrm{RW}}} T'$. As indicated by Lemma 2, $S \xrightarrow{\widehat{\mathrm{RW}}} T'$ can be replaced by a path $S \xrightarrow{\mathrm{RW}} T_1 \xrightarrow{\mathrm{WW}} T_2 \xrightarrow{\mathrm{WW}} \cdots \xrightarrow{\mathrm{WW}} T'$, Therefore, $S' \xrightarrow{\mathrm{SO}\ ;\ \widehat{\mathrm{RW}}} T'$ can be replaced by the path $S' \xrightarrow{\mathrm{SO}\ ;\ \mathrm{RW}} T_1 \xrightarrow{\mathrm{WW}} T_2 \xrightarrow{\mathrm{WW}} \cdots \xrightarrow{\mathrm{WW}} T'$ in $\widehat{\mathcal{G}}'$.
  (3) Consider a WR ; $\widehat{\mathrm{RW}}$ edge in C'. This is similar to case (2).
  (4) Consider a WW ; $\widehat{\mathrm{RW}}$ edge in C'. This is similar to case (2).
  (5) Consider a $S' \xrightarrow{\widehat{\mathrm{WW}}\ ;\ \mathrm{RW}} T'$ edge in C'. Suppose that it is derived from $S' \xrightarrow{\widehat{\mathrm{WW}}} S \xrightarrow{\mathrm{RW}} T'$. The edge $S' \xrightarrow{\widehat{\mathrm{WW}}} S$ can be replaced by a path $S' \xrightarrow{\mathrm{WW}} T_1 \xrightarrow{\mathrm{WW}} T_2 \xrightarrow{\mathrm{WW}} \cdots \xrightarrow{\mathrm{WW}} T_n \xrightarrow{\mathrm{WW}} S$. Therefore, $S' \xrightarrow{\widehat{\mathrm{WW}}\ ;\ \mathrm{RW}} T'$ can be replaced by a path $S' \xrightarrow{\mathrm{WW}} T_1 \xrightarrow{\mathrm{WW}} T_2 \xrightarrow{\mathrm{WW}} \cdots \xrightarrow{\mathrm{WW}} T_n \xrightarrow{\mathrm{WW}\ ;\ \mathrm{RW}} T'$ in $\widehat{\mathcal{G}}'$.
  (6) Consider a $S' \xrightarrow{\widehat{\mathrm{WW}}\ ;\ \widehat{\mathrm{RW}}} T'$ edge in C'. Suppose that it is derived from $S' \xrightarrow{\mathrm{WW}} S \xrightarrow{\mathrm{RW}} T'$. Replace $S' \xrightarrow{\mathrm{WW}} S$ with a path $S' \xrightarrow{\mathrm{WW}} T_1 \xrightarrow{\mathrm{WW}} T_2 \xrightarrow{\mathrm{WW}} \cdots \xrightarrow{\mathrm{WW}} T_n \xrightarrow{\mathrm{WW}} S$. Replace $S \xrightarrow{\mathrm{RW}} T'$ with a path $S \xrightarrow{\mathrm{RW}} T_{n+1} \xrightarrow{\mathrm{WW}} T_{n+2} \xrightarrow{\mathrm{WW}} \cdots \xrightarrow{\mathrm{WW}} T_{n+m} \xrightarrow{\mathrm{WW}} T'$, as indicated by Lemma 2. Therefore, $S' \xrightarrow{\widehat{\mathrm{WW}}\ ;\ \widehat{\mathrm{RW}}} T'$ can be replaced by a path $S' \xrightarrow{\mathrm{WW}} T_1 \xrightarrow{\mathrm{WW}} T_2 \xrightarrow{\mathrm{WW}} \cdots \xrightarrow{\mathrm{WW}} T_n \xrightarrow{\mathrm{WW}\ ;\ \mathrm{RW}} T_{n+1} \xrightarrow{\mathrm{WW}} T_{n+2} \xrightarrow{\mathrm{WW}} \cdots \xrightarrow{\mathrm{WW}} T_{n+m} \xrightarrow{\mathrm{WW}} T'$ in $\widehat{\mathcal{G}}'$.

□

C COMPLEXITY ISSUES

It is known that verifying whether a given general history satisfies SSER, SER or SI is NP-complete [9, 38]. In this section, we show that verifying whether a given mini-transaction history *without unique values* satisfies SSER, SER, or SI is also NP-complete.

Theorem 9. *The problem of verifying whether a given mini-transaction history without unique values satisfies* SSER *is* NP-complete.

Proof. Directly from [23, Theorem 4.11] which shows that verifying whether an execution with read&write operations only satisfies LIN is NP-complete. □

Our proof of the NP-hardness of verifying SER for mini-transaction histories without unique values relies on the NP-hardness of verifying Sequential Consistency (SC) [33]. SC requires that all operations appear to be executed in some sequential order that is consistent with the session order.

Definition 11 (Sequential Consistency [33]). *A history $\mathcal{H}$ is sequentially consistent if and only if there exists a permutation Π of all the operations in $\mathcal{H}$ such that Π preserves the program order of operations and follows the sequential semantics of each object.*

Note that when each transaction comprises only one operation, SER is the same as SC.

Theorem 10. *The problem of verifying whether a given mini-transaction history without unique values satisfies* SER *is* NP-complete.

Proof. Directly from [23, Theorem 4.9] which shows that verifying whether an execution with read&write operations only satisfies SC is NP-complete. □

The proof of the NP-hardness of verifying SI for mini-transaction histories without unique values is much more involved. It is heavily inspired by that of [9, Theorem 3.2].[3] In the axiomatic framework of [9], SI is characterized by two axioms: Prefix and Conflict. The crucial difference is that in histories without unique values, the write-read relation is not given as input. In our context, a mini-transaction history (without unique values) satisfies SI if there exist a write-read relation, denoted by wr to keep the notation consistent with [9], and a commit order co (which is a strict total order on the set of transactions in $\mathcal{H}$) such that the Prefix and Conflict axioms are satisfied. In the following, we omit the RT relation in a history since it is not relevant for SI. An *abstract execution* $\mathcal{A} = (\mathcal{T}, so, wr, co)$ is a history $(\mathcal{T}, so)$ associated with a write-read relation wr and a commit order co. We use R^* to denote the reflexive and transitive closure of the relation R.

Definition 12 (Prefix axiom [9]). *An abstract execution $\mathcal{A} = (\mathcal{T}, \mathrm{SO}, wr, co)$ satisfies the* Prefix *axiom if and only if*

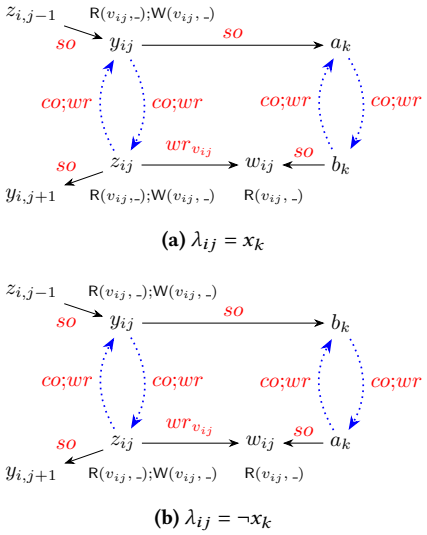
$$\forall x \in \mathrm{X}.\ \forall t_1, t_2, t_3 \in \mathcal{T}.$$
$$t_1 \neq t_2 \wedge \langle t_1, t_3\rangle \in wr(x) \wedge t_2 \vdash \mathrm{W}(x, \_) \wedge \langle t_2, t_3\rangle \in co^*; (wr \cup \mathrm{SO}) \Rightarrow \langle t_2, t_1\rangle \in co$$

Definition 13 (Conflict axiom [9]). *An abstract execution $\mathcal{A} = (\mathcal{T}, \mathrm{SO}, wr, co)$ satisfies the* Conflict *axiom if and only if*

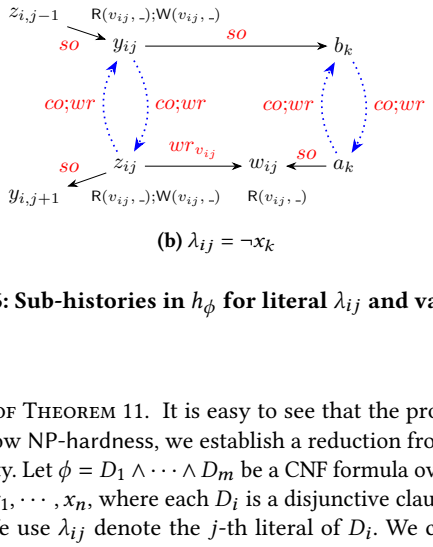
$$\forall x, y \in \mathrm{X}.\ \forall t_1, t_2, t_3, t_4 \in \mathcal{T}.$$
$$t_1 \neq t_2 \wedge \langle t_1, t_3\rangle \in \mathrm{WR}_x \wedge t_2 \vdash \mathrm{W}(x, \_) \wedge t_3 \vdash \mathrm{W}(y, \_) \wedge t_4 \vdash \mathrm{W}(y, \_) \wedge \langle t_2, t_4\rangle \in co^* \wedge \langle t_4, t_3\rangle \in co \Rightarrow \langle t_2, t_1\rangle \in co$$

Theorem 11. *The problem of verifying whether a given mini-transaction history without unique values satisfies* SI *is* NP-complete.

[3] We largely follow the account of [9] and adapt it to our context when necessary.



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



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

Figure 16: Sub-histories in h_ϕ for literal λ_{ij} and variable x_k

Proof of Theorem 11. It is easy to see that the problem is in NP. To show NP-hardness, we establish a reduction from boolean satisfiability. Let $\phi = D_1 \wedge \cdots \wedge D_m$ be a CNF formula over boolean variables $x_1, \cdots, x_n$, where each D_i is a disjunctive clause with m_i literals. We use λ_{ij} denote the j-th literal of D_i. We construct a mini-transaction history h_ϕ for ϕ such that ϕ is satisfiable if and only if h_ϕ satisfies SI.

The insight is to represent truth values of each variable and literal in ϕ with the polarity of the commit order between corresponding transaction pairs. Specifically, for each variable x_k, h_ϕ contains a pair of mini-transaction a_k and b_k such that x_k is false if and only if $\langle a_k, b_k\rangle \in co$. For each literal λ_{ij}, h_ϕ contains a triple of mini-transactions w_{ij}, y_{ij}, and z_{ij} such that λ_{ij} is false if and only if $\langle y_{ij}, z_{ij}\rangle \in co$.

The history h_ϕ should ensure that the co ordering corresponding to an assignment that makes the formula false form a cycle. To this end, we add all pairs $\langle z_{ij}, y_{i,(j+1)\%m_i}\rangle$ in the session order so. Consequently, an unsatisfied clause D_i leads to a cycle of the form $y_{i1} \xrightarrow{co} z_{i1} \xrightarrow{so} y_{i2} \xrightarrow{co} z_{i2} \cdots z_{im_i} \xrightarrow{so} y_{i1}$.

We use special sub-histories to ensure the consistency between the truth value of literals and variables. That is, $\lambda_{ij} = x_k$ is false if and only if x_k is false. Figure 16a shows the sub-history associated to a positive literal $\lambda_{ij} = x_k$, while Figure 16b shows the case of a negative literal $\lambda_{ij} = \neg x_k$.

For a positive literal $\lambda_{ij} = x_k$ (Figure 16a),

(1) we enrich the session order with the pairs $\langle y_{ij}, a_k\rangle$ and $\langle b_k, w_{ij}\rangle$;
(2) we include reads and writes to a variable v_{ij} in the transaction y_{ij} and z_{ij};
(3) we make w_{ij} read v_{ij}; and
(4) we make all the read and written value the same.

The steps (2)–(4) are specially designed for our reduction, different from that in the proof of [9]: *it utilizes the fact that unique values*

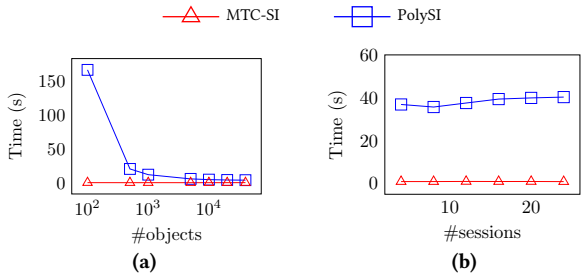


Figure 17: Performance comparison for verifying SI histories.

are not required in such a history and ensures that every transaction in the sub-history is a mini-transaction.

We only need to determine the write-read relation between y_{ij}, z_{ij}, and w_{ij}, since other transactions do not read or write the same variable. Specifically, we construct a write-read relation from z_{ij} to w_{ij}. For transactions y_{ij} and z_{ij}, which read and write the same variable with the same value, we require the direction of wr between y_{ij} and z_{ij} be the same as that of co between them. Otherwise, the history h_ϕ associated with these wr and co relations cannot satisfy Prefix *and* Conflict.

Lemma 5. *The special sub-histories enforce that if history h_ϕ satisfies SI, then there exist a write-read relation wr and a commit order co such that $\langle h_\phi, wr, co\rangle$ satisfies* Prefix *and* Conflict *and*

$$\langle a_k, b_k\rangle \in co \text{ iff } \langle y_{ij}, z_{ij}\rangle \in co \text{ when } \lambda_{ij} = x_k, \text{ and}$$
$$\langle a_k, b_k\rangle \in co \text{ iff } \langle z_{ij}, y_{ij}\rangle \in co \text{ when } \lambda_{ij} = \neg x_k.$$

The proof of Lemma 5 and the remaining correctness proof of the reduction can be conducted similarly as those in [9]. We refer the interested reader to [9] for details.

□

D ADDITIONAL RESULTS FOR VERIFYING SI

MTC-SI surpasses PolySI for verifying SI histories with varying objects and sessions, as shown in Figure 17.

E END-TO-END CHECKING PERFORMANCE FOR SI

MTC-SI with the MT workload generation substantially outperforms PolySI with the GT workload generation under varying levels of concurrency, as shown in Figure 18.

F ISOLATION BUGS REDISCOVERED BY MTC

Figure 19 depicts the counterexamples reported by MTC when checking six releases of five DBMSs (see also Table 1).

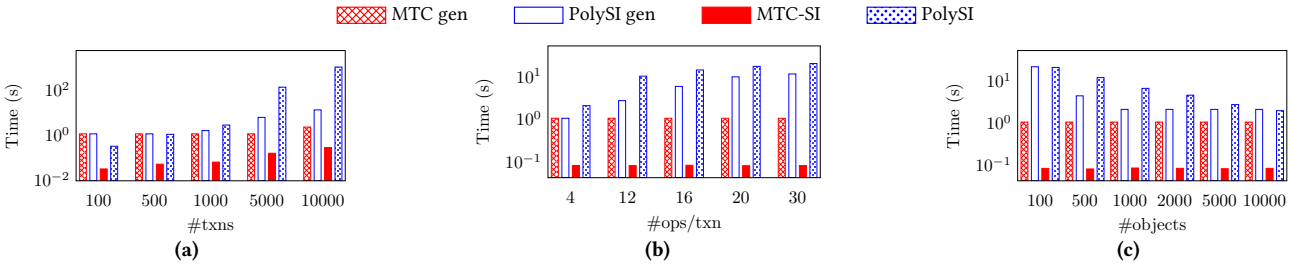
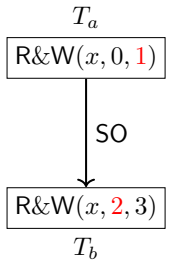
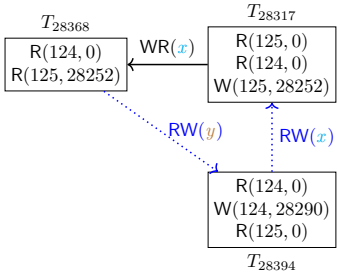


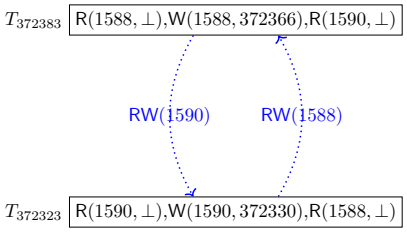
Figure 18: End-to-end checking time for SI, decomposed for the history generation and verification stages.



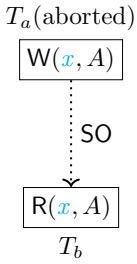
(a) Cassandra: READ UNCOMMITTED that violates SSER



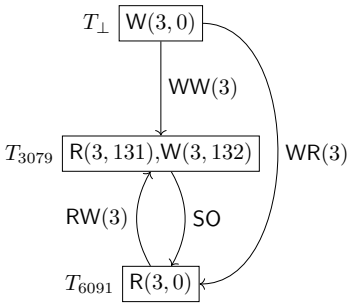
(b) PostgreSQL: WRITE SKEW that violates SER



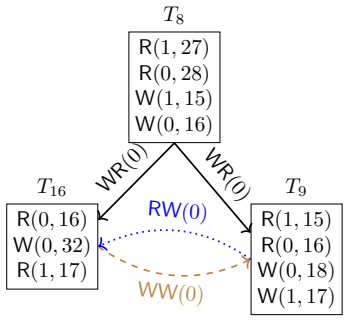
(c) PostgreSQL: LONG FORK that violates SER



(d) MongoDB: READ UNCOMMITTED that violates SI



(e) Dgraph: CAUSALITY VIOLATION that violates SI



(f) MariaDB Galera: LOST UPDATE that violates SI

Figure 19: Rediscovered isolation bugs by MTC.