Deciding Robustness for Lower SQL Isolation Levels

BAS KETSMAN, Vrije Universiteit Brussel, Belgium
CHRISTOPH KOCH, École Polytechnique Fédérale de Lausanne, Switzerland
FRANK NEVEN and BRECHT VANDEVOORT, UHasselt, Data Science Institute, ACSL, Belgium

While serializability always guarantees application correctness, lower isolation levels can be chosen to improve transaction throughput at the risk of introducing certain anomalies. A set of transactions is robust against a given isolation level if every possible interleaving of the transactions under the specified isolation level is serializable. Robustness therefore always guarantees application correctness with the performance benefit of the lower isolation level. While the robustness problem has received considerable attention in the literature, only sufficient conditions have been obtained. The most notable exception is the seminal work by Fekete where he obtained a characterization for deciding robustness against SNAPSHOT ISOLATION. In this article, we address the robustness problem for the lower SQL isolation levels READ UNCOMMITTED and READ COMMITTED, which are defined in terms of the forbidden dirty write and dirty read patterns. The first main contribution of this article is that we characterize robustness against both isolation levels in terms of the absence of counter-example schedules of a specific form (split and multi-split schedules) and by the absence of cycles in interference graphs that satisfy various properties. A critical difference with Fekete's work, is that the properties of cycles obtained in this article have to take the relative ordering of operations within transactions into account as READ UNCOMMITTED and READ COMMITTED do not satisfy the atomic visibility requirement. A particular consequence is that the latter renders the robustness problem against READ COMMITTED coNP-complete. The second main contribution of this article is the coNP-hardness proof. For READ UNCOMMITTED, we obtain LOGSPACE-completeness.

CCS Concepts: • **Information systems** → **Database transaction processing**; • **Theory of computation** → *Database theory;*

Additional Key Words and Phrases: Concurrency control, SQL isolation levels

ACM Reference format:
Bas Ketsman, Christoph Koch, Frank Neven, and Brecht Vandevoort. 2022. Deciding Robustness for Lower SQL Isolation Levels. *ACM Trans. Database Syst.* 47, 4, Article 13 (November 2022), 41 pages.
https://doi.org/10.1145/3561049

This work is funded by FWO-grant G019921N.

Authors' addresses: B. Ketsman, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium; email: bas.ketsman@vub.be, C. Koch, EPFL IC IINFCOM DATA, BC 260, Station 14, CH-1015 Lausanne; email: christoph.koch@epfl.ch; F. Neven and B. Vandevoort (corresponding author), UHasselt, Data Science Institute, ACSL, Agoralaan gebouw D, 3590 Diepenbeek, Belgium; emails: {frank.neven, brecht.vandevoort}@uhasselt.be.

0362-5915/2022/11-ART13 $15.00
https://doi.org/10.1145/3561049

1 INTRODUCTION

To guarantee consistency during concurrent execution of transactions, most database management systems offer a serializable isolation level. Serializability ensures that the effect of concurrent execution of transactions is always equivalent to a serial execution where transactions are executed in sequence one after another. The database system thereby guarantees perfect isolation for every transaction. For application programmers, perfect isolation is extremely important, as it implies that they only need to reason about correctness properties of individual transactions. Ensuring serializability, however, comes at a non-trivial performance cost [24]. Database systems therefore provide the ability to trade off isolation guarantees for improved performance by offering a variety of isolation levels. Even though isolation levels lower than serializability are often configured by default (see, e.g., Reference [5]), executing transactions concurrently under such isolation levels is not without risk as it can introduce certain anomalies. Sometimes, however, a set of transactions can be executed at an isolation level lower than serializability without introducing any anomalies. This is, for instance, the case for the TPC-C benchmark application [21] running under SNAPSHOT ISOLATION. In such a case, the set of transactions is said to be *robust* against a particular isolation level. More formally, *a set of transactions is robust against a given isolation level if every possible interleaving of the transactions allowed under the specified isolation level is serializable.* Detecting robustness is highly desirable as it allows to guarantee perfect isolation at the performance cost of a lower isolation level.

Fekete et al. [16] initiated the study of robustness in the context of SNAPSHOT ISOLATION, referring to it as the *acceptability* problem and providing a sufficient condition in terms of the absence of cycles with specific types of edges in the static dependency graph (what we and Fekete [15] call interference graph). This result was extended by Bernardi and Gotsman [10] by providing sufficient conditions for deciding robustness against the different isolation levels that can be defined in a declarative framework as introduced by Cerone et al. [11]. This framework provides a uniform specification of various isolation levels (including SNAPSHOT ISOLATION) that admit atomic visibility, a condition requiring that either all or none of the updates of each transaction are visible to other transactions. The atomic visibility assumption is key as it allows to specify isolation levels by consistency axioms on the level of transactions rather than on the granularity of individual operations within each transaction. The sufficient conditions are again based on the absence of cycles with certain types of edges.

In a seminal paper, Fekete [15] obtained a characterization for deciding robustness against SNAPSHOT ISOLATION, which should be contrasted with the work mentioned above that only provide sufficient conditions. In this article, we extend the former work by providing characterizations for robustness against the lower SQL isolation levels READ UNCOMMITTED and READ COMMITTED, which are defined in terms of the forbidden dirty write and dirty read patterns [9]. Especially READ COMMITTED is a very relevant isolation level as it is the default isolation level in quite a number of database systems [6] and also because it is one of the few isolation levels providing highly available transactions [5]. Furthermore, as READ COMMITTED and by extension READ UNCOMMITTED, provide a low performance penalty, establishing robustness against these isolation levels allows rapid concurrent execution while guaranteeing perfect isolation. Alomari and Fekete [3] already studied robustness against READ COMMITTED and provide a sufficient condition that is not a necessary one.

To provide some insight into the technical challenges, we introduce some terminology by example (formal definitions are given in Section 2). As usual, a transaction is a sequence of read and write operations on objects followed by a commit. Consider, for instance, the set of transactions $\mathcal{T} = \{T_1, T_2\}$ with $T_1 = \mathtt{W}_1[\mathtt{x}]\mathtt{R}_1[\mathtt{z}]\mathtt{W}_1[\mathtt{y}]\mathtt{C}_1$ and $T_2 = \mathtt{W}_2[\mathtt{z}]\mathtt{R}_2[\mathtt{y}]\mathtt{W}_2[\mathtt{x}]\mathtt{C}_2$. Here, $\mathtt{W}_\mathtt{i}[\mathtt{x}]$ and $\mathtt{R}_\mathtt{i}[\mathtt{x}]$ denote a read and a write operation to object $\mathtt{x}$ by transaction T_i, whereas $\mathtt{C}_\mathtt{i}$ is the commit operation of T_i. A schedule for $\mathcal{T}$ then is an ordering of all operations occurring in transactions in $\mathcal{T}$. For instance,

$$\begin{array}{lllll}
\text{schedule } s_1: & & \mathtt{W}_1[\mathtt{x}]\mathtt{R}_1[\mathtt{z}]\mathtt{W}_1[\mathtt{y}]\mathtt{C}_1 & & (T_1) \\
& \mathtt{W}_2[\mathtt{z}] & & \mathtt{R}_2[\mathtt{y}]\mathtt{W}_2[\mathtt{x}]\mathtt{C}_2 & (T_2)
\end{array}$$

$$\begin{array}{llllll}
\text{schedule } s_2: & \mathtt{W}_1[\mathtt{x}]\mathtt{R}_1[\mathtt{z}] & & \mathtt{W}_1[\mathtt{y}]\mathtt{C}_1 & & (T_1) \\
& & \mathtt{W}_2[\mathtt{z}]\mathtt{R}_2[\mathtt{y}] & & \mathtt{W}_2[\mathtt{x}]\mathtt{C}_2 & (T_2)
\end{array}$$

Fig. 1. Schedules s_1 and s_2 for $\mathcal{T} = \{T_1, T_2\}$.

s_1 and s_2 as displayed in Figure 1 are schedules for $\mathcal{T}$. A schedule is not allowed under isolation level READ UNCOMMITTED when it exhibits a dirty write: a pattern of the form $\mathtt{W}_1[\mathtt{x}] \cdots \mathtt{W}_2[\mathtt{x}] \cdots \mathtt{C}_1$, that is, T_2 writes to an object that has been modified by a transaction T_1 that has not yet committed. Both s_1 and s_2 are allowed under READ UNCOMMITTED. The isolation level READ COMMITTED prohibits dirty writes as well as dirty reads. The latter is a pattern of the form $\mathtt{W}_2[\mathtt{z}] \cdots \mathtt{R}_1[\mathtt{z}] \cdots \mathtt{C}_2$. That is, T_1 reads an object that has been modified by a transaction T_2 that has not yet committed. The schedule s_1 is not allowed under READ COMMITTED. Notice that s_1 and s_2 are not conflict serializable as their conflict graphs admit a cycle.[1] Indeed, consider s_1, $\mathtt{W}_2[\mathtt{z}]$ occurring before $\mathtt{R}_1[\mathtt{z}]$ in s_1 implies that in any conflict equivalent sequential schedule T_2 should occur before T_1, while $\mathtt{W}_1[\mathtt{x}]$ occurring before $\mathtt{W}_2[\mathtt{x}]$ in s_1 implies the converse.

We start by studying robustness against READ UNCOMMITTED. This means that for a given set of transactions, we need to check whether there is a counter-example schedule that is allowed under READ UNCOMMITTED and that is not serializable, that is, contains a cycle in its conflict graph. Notice that for $\mathcal{T} = \{T_1, T_2\}$ as defined above s_1 constitutes such a counter example. Furthermore, s_1 is of a very particular form. Indeed, s_1 can be seen as the schedule constructed by splitting T_2 into two parts ($\mathtt{W}_2[\mathtt{z}]$ and $\mathtt{R}_2[\mathtt{y}]\mathtt{W}_2[\mathtt{x}]\mathtt{C}_2$) and placing the complete transaction T_1 in between. We call such schedules a *split schedule*. They can also be defined for sets of transactions consisting of more than two transactions by splitting one transaction in two parts and placing all other transaction in between (cf. Figure 2). We show that the existence of a counter-example schedule that has the form of a split schedule provides a necessary and sufficient condition for deciding robustness against READ UNCOMMITTED.

Fekete [15] introduced the notion of an interference graph for a set of transactions and obtained a characterization for deciding robustness against SNAPSHOT ISOLATION in terms of the absence of a cycle with certain types of edges. We mimic his result by obtaining an additional characterization of deciding robustness against READ UNCOMMITTED in terms of the absence of cycles in the interference graphs that are prefix-write-conflict-free.[2] It is important to point out the main difference with the work of Fekete: SNAPSHOT ISOLATION admits atomic visibility implying that cycles in the interference graph can refer to the global ordering of transactions and can ignore the ordering of operations within transactions. For READ UNCOMMITTED, we cannot rely on atomic visibility and need to take into account the specific conflicting operations that generate the edges in the interference graph. In addition, the notion of prefix-write-conflict-free cycle requires to isolate a single transaction (the one witnessing transferability, see Section 3, and the one that will be split in the counter-example schedule) and determine non-existence of write-conflicts with respect to a prefix of this transaction (so the order of operations matters). That being said, the complexity of testing robustness against READ UNCOMMITTED can be done very efficiently, as we show it to be LOGSPACE-complete.

[1] See Section 2.2 for a definition of conflict graphs and how acyclity implies serializability.
[2] See Section 4 for a formal definition.

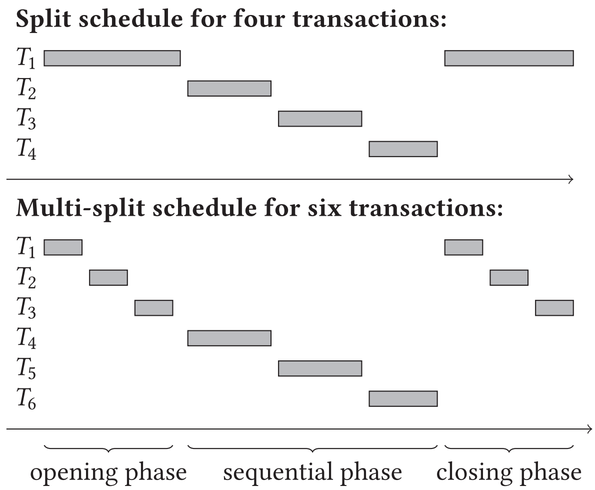


Fig. 2. Abstract presentation of split and multi-split schedule. The drawing omits a possible trailing sequence of non-interleaved transactions (cf. Definition 8 and Definitions 18).

Next, we turn to robustness against READ COMMITTED. Schedule s_2 shown in Figure 1 is allowed under READ COMMITTED and is not serializable. It is hence a counter example showing that $\mathcal{T}$ is not robust against READ COMMITTED. Notice that s_2 is not a split schedule. In fact, it can be argued that there is no split schedule for $\mathcal{T}$ that is allowed under READ COMMITTED. This means that the existence of a counter-example schedule in the form of a split schedule is not a necessary condition for deciding robustness against READ COMMITTED. We show that counter examples do not need to take arbitrary forms either. We obtain a characterization for deciding robustness against READ COMMITTED in terms of counter-example schedules that take the form of multi-split schedules as illustrated in Figure 2. In contrast to a split schedule where one transaction is split open and all other transactions are inserted, a multi-split schedule can open several such transactions but needs to close them in the same order.

We obtain an equivalent characterization in terms of the absence of a multi-prefix-conflict-free cycle in the interference graph. The latter is a rather involved property of cycles that much more than the notion of prefix-write-conflict-free mentioned previously depends on the ordering of operations within transactions. Using this notion, we show that deciding robustness against READ COMMITTED is CONP-complete. The lower bound proof is a rather involved reduction from 3SAT that bears on ideas from the NP-hardness proof for the PROPERLYCOLOREDCYCLE problem discussed in Section 5.2. The latter should be contrasted with robustness against SNAPSHOT ISOLATION for which the algorithm in Reference [15] implies a PTIME upper bound.

Following the work of Fekete [15], we are the first to obtain a complete characterization for robustness against the considered isolation levels. The main contributions of this article can be summarized as follows:

(1) providing characterizations for deciding robustness against READ UNCOMMITTED and READ UNCOMMITTED in terms of the absence of (i) counter-example schedules of various shapes and (ii) cycles in interference graphs of various forms; these characterizations provide direct upper bounds on the complexity of deciding robustness; and.
(2) CONP-hardness of deciding robustness against READ COMMITTED.

Outline. We introduce the necessary definitions in Section 2. We introduce key notions in Section 3 in the context of robustness against no isolation level. We consider robustness against READ

UNCOMMITTED and READ COMMITTED in Sections 4 and 5, respectively. We discuss related work in Section 7 and conclude in Section 8.

Novelty Requirement. The present article is the full version of Reference [17] and supplies all proofs. In particular, full proofs of the following non-trivial results are added: Theorems 12, 16, 17, and Lemma 32. In addition, we added novel material on robustness for schedules with missing and repeating transactions in Section 6 that was not present in the conference version.

2 DEFINITIONS

2.1 Transactions and Schedules

For natural numbers i and j with $i \leq j$, denote by $[i, j]$ the set $\{i, \ldots, j\}$. We fix an infinite set of objects **Obj**. For an object $\mathtt{x} \in \mathbf{Obj}$, we denote by $\mathtt{R[x]}$ a *read* operation on $\mathtt{x}$ and by $\mathtt{W[x]}$ a *write* operation on $\mathtt{x}$. We also assume a special *commit* operation denoted by $\mathtt{C}$. A *transaction* T over **Obj** is a sequence of read and write operations on objects in **Obj** followed by a commit. In the sequel, we leave the set of objects **Obj** implicit when it is clear from the context and just say transaction rather than transaction over **Obj**. We also sometimes just say *reads* and *writes* rather than read and write operations.

We assume that a transaction performs at most one write and at most one read per object. The latter is a common assumption (see, e.g., Reference [15]) and is made to simplify exposition; all our results carry over to the more general setting in which multiple writes and reads per object are allowed.

Formally, we model a transaction as a linear order $(T, \leq_T)$, where T is the set of (read, write, and commit) operations occurring in the transaction and $\leq_T$ encodes the ordering of the operations. As usual, we use $<_T$ to denote the strict ordering.

For an operation $b \in T$, we denote by $\text{prefix}_b(T)$ the restriction of T to all operations that are smaller than or equal to b according to $\leq_T$. Similarly, we denote by $\text{postfix}_b(T)$ the restriction of T to all operations that are strictly larger than b according to $\leq_T$. Throughout the article, we interchangeably consider transactions both as linear orders as well as sequences. Therefore, T is then equal to the sequence $\text{prefix}_b(T)$ followed by $\text{postfix}_b(T)$, which we denote by $\text{prefix}_b(T) \cdot \text{postfix}_b(T)$ for every $b \in T$.

When considering a set $\mathcal{T}$ of transactions, we assume that every transaction in the set has a unique id i and write T_i to make this id explicit. Similarly, to distinguish the operations from different transactions, we add this id as index to the operation. That is, we write $\mathtt{W_i[x]}$ and $\mathtt{R_i[x]}$ to denote a write and read on object x occurring in transaction T_i; similarly $\mathtt{C_i}$ denotes the commit operation in transaction T_i. Notice that this convention is consistent with the literature (see, e.g., References [9, 15]).

A *schedule* s over a set $\mathcal{T}$ of transactions is a sequence of all the operations occurring in transactions in $\mathcal{T}$ in which the order of operations from the different transactions is consistent with their order in the respective transactions. Formally, we model a schedule as a linear order $(s, \leq_s)$ where s is the set containing all operations of transactions in $\mathcal{T}$ and $\leq_s$ encodes the ordering of these operations with the additional constraint that $a <_T b$ implies $a <_s b$ for every $T \in \mathcal{T}$ and every $a, b \in T$.

The absence of aborts in our definition of schedule is consistent with the common assumption [10, 15] that an underlying recovery mechanism will rollback transactions that interfere with aborted transactions.

A schedule s over a set of transactions $\mathcal{T}$ is *sequential* if its transactions are not interleaved with operations from other transactions. That is, for every $a, b, c \in s$ with $a <_s b <_s c$ and $a, c \in T$ implies $b \in T$ for every $T \in \mathcal{T}$. Adopting the view of schedules as sequences, the schedule

$s_1 = T_1 \cdot T_2 \cdot \ldots \cdot T_n$ is an example of a sequential schedule for the set of transactions $\{T_1, T_2, \ldots, T_n\}$ as is any permutation of transactions in s_1.

2.2 Conflict Serializability

We say that two operations a_i and b_j from different transactions T_i and T_j are *conflicting* if both are operations on the same object, and at least one of them is a write. That is, $\mathtt{R_i[x]}$ and $\mathtt{W_j[x]}$ and $\mathtt{W_i[x]}$ and $\mathtt{W_j[x]}$ are conflicting operations, while $\mathtt{R_i[x]}$ and $\mathtt{R_j[x]}$ are not. Furthermore, a commit operation never conflicts with any other operation. Two schedules s and s' are *conflict equivalent* if they are over the same set $\mathcal{T}$ of transactions and if any pair of conflicting operations a and b is ordered the same in both, that is, $a \leq_s b$ iff $a \leq_{s'} b$.

Definition 1. A schedule s is *conflict serializable* if it is conflict equivalent to a sequential schedule.

A *conflict graph* $CG(s)$ for schedule s over a set of transactions $\mathcal{T}$ is defined as usual [18]: It is the graph whose nodes are the transactions in $\mathcal{T}$ and where there is an edge from T_i to T_j if T_i has an operation b_i that conflicts with an operation a_j in T_j with $b_i <_s a_j$.[3] Since we are usually interested not only in the existence of conflicting operations but also in the operations themselves, we assume the existence of a labeling function λ mapping each edge to a set of pairs of operations. Formally, $(b_i, a_j) \in \lambda(T_i, T_j)$ iff the operation $b_i \in T_i$ conflicts with the operation $a_j \in T_j$ and $b_i <_s a_j$. For ease of notation, we choose to represent $CG(s)$ as a set of quadruples (T_i, b_i, a_j, T_j) denoting all possible pairs of these transactions T_i and T_j with all possible choices of conflicting operations b_i and a_j. Henceforth, we refer to these quadruples simply as edges. Notice that edges only contain read and write operations, no commit operations.

A *cycle* C in $CG(s)$ is a non-empty sequence of edges

$$(T_1, b_1, a_2, T_2), (T_2, b_2, a_3, T_3), \ldots, (T_n, b_n, a_1, T_1)$$

in $CG(s)$, in which every transaction is mentioned exactly twice. Note that cycles are by definition simple. Here, transaction T_1 starts and concludes the cycle. For a transaction T_i in C, we denote by $C[T_i]$ the cycle obtained from C by letting T_i start and conclude the cycle while otherwise respecting the order of transactions in C. That is, $C[T_i]$ is the sequence

$$(T_i, b_i, a_{i+1}, T_{i+1}) \cdots (T_n, b_n, a_1, T_1)(T_1, b_1, a_2, T_2) \cdots (T_{i-1}, b_{i-1}, a_i, T_i).$$

We recall the following well-known result:

THEOREM 2 ([18]). *A schedule s is conflict serializable iff the conflict graph for s is acyclic.*

2.3 Isolation Levels

We define isolation levels in terms of the concurrency phenomena that we want to exclude from schedules [9].

Let s be a schedule for a set $\mathcal{T}$ of transactions.

- Then, *s exhibits a dirty write* iff there are two different transactions T_i and T_j in $\mathcal{T}$ and an object $\mathtt{x}$ such that

$$\mathtt{W_i[x]} <_s \mathtt{W_j[x]} <_s \mathtt{C_i}.$$

  That is, transaction T_j writes to an object that has been modified earlier by T_i, but T_i has not yet issued a commit.

[3] Throughout the article, we adopt the following convention: A b operation can be understood as a "before" while an a can be interpreted as an "after."

- Furthermore, s *exhibits a dirty read* iff there are two different transactions T_i and T_j in $\mathcal{T}$ and an object $\mathtt{x}$ such that

$$\mathtt{W_i[x]} <_s \mathtt{R_j[x]} <_s \mathtt{C_i}.$$

  That is, transaction T_j reads an object that has been modified earlier by T_i, but T_i has not yet issued a commit.

Definition 3. A schedule is *allowed under isolation level* READ UNCOMMITTED if it exhibits no dirty writes, and it is *allowed under isolation level* READ COMMITTED if, in addition, it also exhibits no dirty reads. For convenience, we use the term NO ISOLATION to refer to the isolation level that allows all schedules.

Notice that every schedule is allowed under NO ISOLATION. Furthermore, every schedule allowed under READ COMMITTED is also allowed under READ UNCOMMITTED. It is accustomed to view an isolation level as a set of allowed schedules [18].

We say that an isolation level $\mathcal{I}$ is a *restriction* of an isolation level $\mathcal{I}'$, denoted $\mathcal{I} \subseteq \mathcal{I}'$, if the fact that a schedule s is allowed under $\mathcal{I}$ implies that s is allowed under $\mathcal{I}'$ as well.

2.4 Robustness

Next, we define the robustness property [10] (also called *acceptability* in References [15, 16]), which guarantees serializability for all schedules of a given set of transactions for a given isolation level.

Definition 4 (Robustness). A set $\mathcal{T}$ of transactions is *robust* against an isolation level if every schedule for $\mathcal{T}$ that is allowed under that isolation level is conflict serializable.

For an isolation level $\mathcal{I}$, ROBUSTNESS($\mathcal{I}$) is the problem to decide if a given set of transactions $\mathcal{T}$ is robust against $\mathcal{I}$. The following is an immediate observation:

LEMMA 5. *Let $\mathcal{T}$ be a set of transactions. Let $\mathcal{I}$ and $\mathcal{I}'$ be isolation levels with $\mathcal{I} \subseteq \mathcal{I}'$. Then $\mathcal{T}$ is robust against $\mathcal{I}'$ implies that $\mathcal{T}$ is robust against $\mathcal{I}$.*

PROOF. Indeed, $\mathcal{T}$ is robust against $\mathcal{I}'$ means that there is no schedule s that is allowed under $\mathcal{I}'$ for which $CG(s)$ is acyclic. As $\mathcal{I} \subseteq \mathcal{I}'$, there is also no such schedule s that is allowed under $\mathcal{I}$, which in turn implies that $\mathcal{T}$ is robust against $\mathcal{I}$. □

3 NO ISOLATION LEVEL

We start by studying the toy isolation level NO ISOLATION that admits all schedules. The present section serves as a warm up for the remainder of the article and allows to discuss key notions like the interference graph, transferable cycle, and split schedule in a simplified setting.

We use a variant of the interference graph, as introduced by Fekete [15], which essentially lifts the notion of a conflict graph from schedules to sets of transactions. Consistent with our definition of conflict graph, we expose conflicting operations via an explicit labeling of edges.

Definition 6. For a set of transactions $\mathcal{T}$, the *interference graph* $IG(\mathcal{T})$ for $\mathcal{T}$ is the graph whose nodes are the transactions in $\mathcal{T}$ and where there is an edge from T_i to T_j if there is an operation in T_i that conflicts with some operation in T_j. Again, we assume a labeling function λ mapping each edge to a set of pairs of conflicting operations. Formally, $(b_i, a_j) \in \lambda(T_i, T_j)$ iff there is an operation $b_i \in T_i$ that conflicts with an operation $a_j \in T_j$.

For convenience, just like for conflict graphs, we choose to represent $IG(\mathcal{T})$ as a set of quadruples of the form (T_i, b_i, a_j, T_j). That is, $(T_i, b_i, a_j, T_j) \in IG(\mathcal{T})$ iff there is an edge (T_i, T_j) and $(b_i, a_j) \in \lambda(T_i, T_j)$. Again, we then refer to these quadruples simply as edges.

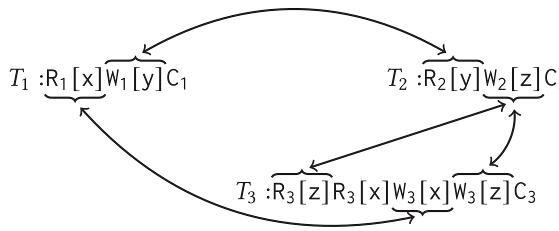


Fig. 3. $IG(\mathcal{T})$ for $\mathcal{T} = \{T_1, T_2, T_3\}$ as defined in Example 9.

Notice that $(T_i, b_i, a_j, T_j) \in IG(\mathcal{T})$ implies $(T_j, a_j, b_i, T_i) \in IG(\mathcal{T})$. Furthermore, the conflict graph $CG(s)$ for a schedule s for $\mathcal{T}$ is always a subgraph of the interference graph $IG(\mathcal{T})$ for $\mathcal{T}$. Therefore, every cycle in $CG(s)$ is a cycle in $IG(\mathcal{T})$. However, the converse is not always true. Sometimes a cycle in $IG(\mathcal{T})$ can be found that does not translate to a corresponding cycle in the conflict graph for any schedule for $\mathcal{T}$. We therefore introduce the notion of a *transferable cycle* in an interference graph and show in Lemma 10 that whenever there is a transferable cycle in $IG(\mathcal{T})$ there is a schedule s of a specific form called a split schedule (as defined in Definition 8) that admits a cycle in $CG(s)$.

Definition 7. Let $\mathcal{T}$ be a set of transactions and C a cycle in $IG(\mathcal{T})$. Then, C is *non-trivial* if for some pair of edges (T_i, b_i, a_j, T_j) and (T_j, b_j, a_k, T_k) in C the operations b_j and a_j are different. Furthermore, C is *transferable* if $b_j <_{T_j} a_j$ for some pair of edges (T_i, b_i, a_j, T_j) and (T_j, b_j, a_k, T_k) in C. We then say that C is transferable in T_j on operations (b_j, a_j).

When a cycle is transferable in T on (b, a), we create a split schedule by splitting T between b and a, inserting all other transactions from the cycle in the created opening while maintaining their ordering and appending at the end all other transactions not occurring in the cycle in an arbitrary order. Notice that split schedules exhibit a cycle in their conflict graph. Split schedules are formally defined as follows:

Definition 8 (Split Schedule). Let $\mathcal{T}$ be a set of transactions and C a transferable cycle in $IG(\mathcal{T})$. A *split schedule* for $\mathcal{T}$ based on C has the form

$$\mathsf{prefix}_b(T_1) \cdot T_2 \cdot \ldots \cdot T_m \cdot \mathsf{postfix}_b(T_1) \cdot T_{m+1} \cdot \ldots \cdot T_n,$$

where

- (T_m, b_m, a, T_1) and (T_1, b, a_2, T_2) is a pair of edges in C and C is transferable in T on (b, a);
- $T_1, \ldots, T_m$ are the transactions in $C[T_1]$ in the order as they occur; and,
- $T_{m+1}, \ldots, T_n$ are the remaining transactions in $\mathcal{T}$ in an arbitrary order.

More specifically, we say that the above schedule is a split schedule for $\mathcal{T}$ based on C, T_1, and b.

We say that a schedule s is a split schedule for $\mathcal{T}$ if there is a transferable cycle C in $IG(\mathcal{T})$ such that s is a split schedule for $\mathcal{T}$ based on C. Figure 2 provides an abstract view of a split schedule omitting the trailing sequence $T_{m+1} \cdots T_n$.

Example 9. Consider $\mathcal{T} = \{T_1, T_2, T_3\}$ with $T_1 = \mathtt{R}_1[\mathtt{x}]\mathtt{W}_1[\mathtt{y}]\mathtt{C}_1$, $T_2 = \mathtt{R}_2[\mathtt{y}]\mathtt{W}_2[\mathtt{z}]\mathtt{C}_2$ and $T_3 = \mathtt{R}_3[\mathtt{z}]\mathtt{R}_3[\mathtt{x}]\mathtt{W}_3[\mathtt{x}]\mathtt{W}_3[\mathtt{z}]\mathtt{C}_3$. Then $IG(\mathcal{T})$ is depicted in Figure 3. The cycle C_1, consisting of the following edges:

$$(T_1, \mathtt{W}_1[\mathtt{y}], \mathtt{R}_2[\mathtt{y}], T_2), (T_2, \mathtt{W}_2[\mathtt{z}], \mathtt{W}_3[\mathtt{z}], T_3), (T_3, \mathtt{W}_3[\mathtt{x}], \mathtt{R}_1[\mathtt{x}], T_1),$$

is transferable in T_3 on $(\mathtt{W}_3[\mathtt{x}], \mathtt{W}_3[\mathtt{z}])$ as $\mathtt{W}_3[\mathtt{x}] <_{T_3} \mathtt{W}_3[\mathtt{z}]$. The cycle C_2, consisting of the following edges:

$$(T_1, \mathtt{W}_1[\mathtt{y}], \mathtt{R}_2[\mathtt{y}], T_2), (T_2, \mathtt{W}_2[\mathtt{z}], \mathtt{R}_3[\mathtt{z}], T_3), (T_3, \mathtt{W}_3[\mathtt{x}], \mathtt{R}_1[\mathtt{x}], T_1),$$

is not transferable in T_3 on $(\mathtt{W}_3[\mathtt{x}], \mathtt{R}_3[\mathtt{z}])$ as $\mathtt{W}_3[\mathtt{x}] \not<_{T_3} \mathtt{R}_3[\mathtt{z}]$. The split schedule s_1 for $\mathcal{T}$ based on C_1, T_3, and $\mathtt{W}_3[\mathtt{x}]$ is as follows:

$$\underbrace{\mathtt{R}_3[\mathtt{z}]\mathtt{R}_3[\mathtt{x}]\mathtt{W}_3[\mathtt{x}]}_{\text{prefix}_b(T_3)}\ \underbrace{\mathtt{R}_1[\mathtt{x}]\mathtt{W}_1[\mathtt{y}]\mathtt{C}_1}_{T_1}\ \underbrace{\mathtt{R}_2[\mathtt{y}]\mathtt{W}_2[\mathtt{z}]\mathtt{C}_2}_{T_2}\ \underbrace{\mathtt{W}_3[\mathtt{z}]\mathtt{C}_3}_{\text{postfix}_b(T_3)},$$

with $b = \mathtt{W}_3[\mathtt{x}]$.

The following lemma collects some interesting properties of transactions.

Lemma 10. *Let $\mathcal{T}$ be a set of transactions.*

(1) *If a schedule s for $\mathcal{T}$ has a cycle C in its conflict graph, then C is a transferable cycle in $IG(\mathcal{T})$.*
(2) *If there is a non-trivial cycle C in $IG(\mathcal{T})$, then there is a transferable cycle C' in $IG(\mathcal{T})$.*
(3) *Let s be a split schedule for $\mathcal{T}$ based on a transferable cycle C in $IG(\mathcal{T})$. Then C is a cycle in $CG(s)$.*

Proof. (1) It follows that C is a cycle in $IG(\mathcal{T})$. Now assume toward a contradiction that C is not transferable, and thus that for every pair of edges (T_i, b_i, a_j, T_j), (T_j, b_j, a_k, T_k) in C operation a_j precedes or equals operation b_j. But then, as C is a cycle and by the definition of a conflict graph, we have that $a_j \leq_{T_j} b_j <_s a_k \leq_{T_k} b_k <_s \cdots <_s a_j$, which implies $a_j <_s a_j$, leading to the desired contradiction.

(2) Let (T_i, b_i, a_j, T_j) and (T_j, b_j, a_k, T_k) be edges in C with $b_j \neq a_j$. If $b_j <_{T_j} a_j$, then C is transferable itself and take C' as C. If $b_j >_{T_j} a_j$, then recall that $IG(\mathcal{T})$ is bidirectional and define C' as the cycle obtained from C by starting in transaction T_j and walking through C against the orientation of its edges. Clearly, C' is transferable.

(3) Follows immediately from the definition of a split schedule. □

We are now ready to formulate a theorem that provides a characterization for deciding robustness against no isolation:

Theorem 11. *Let $\mathcal{T}$ be a set of transactions. The following are equivalent:*

(1) *$\mathcal{T}$ is not robust against isolation level* no isolation*;*
(2) *$IG(\mathcal{T})$ contains a non-trivial cycle; and,*
(3) *there is split schedule s for $\mathcal{T}$.*

Proof. *(1 → 2)* Let s be a schedule for $\mathcal{T}$ that is not conflict serializable. Then there is a cycle C in its conflict graph $CG(\mathcal{T})$ (by Theorem 2), which is a transferable cycle in $IG(\mathcal{T})$ due to Lemma 10(1). Furthermore, a transferable cycle is non-trivial by definition.

(2 → 3) By Lemma 10(2) there is a transferable cycle C in $IG(\mathcal{T})$. This cycle can be used to construct a split schedule for $\mathcal{T}$.

(3 → 1) Immediate by Lemma 10(3). □

Next, we discuss the complexity of deciding robustness. Because the interference graph $IG(\mathcal{T})$ of a set $\mathcal{T}$ of transactions is bidirectional, it has a natural undirected interpretation, which we denote by $IG_u(\mathcal{T})$. Formally, the undirected edge $\{T_i, T_j\}$ occurs in $IG_u(\mathcal{T})$ iff there is an edge $(T_i, b_i, a_j, T_j) \in IG(\mathcal{T})$. In the next theorem, the upper bound is based on the result that undirected reachability is in logspace [19]. The lower bound is by an fo-reduction from the logspace-complete undirected acyclicity problem [14].

Theorem 12. robustness*(*no isolation*) is* logspace*-complete.*

Proof. Given a set of transactions $\mathcal{T}$, the algorithm works as follows. We check, for every pair of incident edges $\{T_i, T_j\}$ and $\{T_j, T_k\}$ in $IG_u(\mathcal{T})$, that the shared end-point T_j witnesses non-triviality and that *either*, T_i and T_k are the same transaction or that T_i and T_k are reachable through a path that omits T_j. The latter can be rephrased as a reachability test for T_i and T_k in the subgraph G_{T_j} of $IG_u(\mathcal{T})$ containing all edges of G except those with T_j as an end-point. If all these checks fail, then $\mathcal{T}$ is robust; otherwise, it is not robust (due to Theorem 11).

The enumeration and the first part of the check can be done straightforwardly in LOGSPACE. For the reachability check, we rely on the famous result by Reingold [19] that *undirected reachability* is LOGSPACE-complete. Of course we do not materialize G_{T_j} (as its materialization would require more than logarithmic space). Instead, we apply Reingold's algorithm and every time this algorithm accesses an edge in G_{T_j}, we test whether this edge exists based on $\mathcal{T}$ and T_j. Note in particular that, given $\mathcal{T}$ and T_j, it is possible to enumerate the nodes and edges of G_{T_j} using logarithmic working space (in other words, we redo the transformation—and reuse its memory—every time an edge in G_{T_j} is accessed [4]).

The lower bound is by an FO-reduction from the LOGSPACE-complete undirected acyclicity problem [14] to transaction robustness against NO ISOLATION.

For the construction, let G be an undirected graph given as input to the acyclicity problem. To formulate an FO-reduction, we assume that graph G is encoded as a predicate E that expresses the edge relation. We assume also an ordering $<_G$ over the nodes in G (which is, for example, derived from G's encoding on the input tape). The goal of our reduction is to transform predicate E into a predicate Opp that encodes a set $\mathcal{T}$ of transactions given as input to the robustness problem.

More precisely, Opp defines the operations in transactions in $\mathcal{T}$ encoded as triples $\mathrm{Opp}(n, a, b)$, representing a write (if $n = a$) or a read (if $n = b$) by transaction T_n to object $x_{a,b}$. In other words, the edges in graph G have become objects and its nodes have become transactions.

Formally, we have the following transformation:

$$\mathrm{Opp} := \{(a, a, b) \mid \mathsf{E}(a, b) \wedge a <_G b\} \cup \{(b, a, b) \mid \mathsf{E}(b, a) \wedge a <_G b\}.$$

Inequality $a <_G b$ ensures that transactions occuring as adjacent nodes in G write to/read from exactly one common object. We assume that operations within a transaction occur in the order as defined by the lexicographical order based on $<_G$ over the pairs (b, a) representing the object that they write to. Notice that by construction, each schedule s over $\mathcal{T}$ is allowed not only under NO ISOLATION but also under READ UNCOMMITTED, because no object occurs in two writes (and therefore s cannot exhibit a dirty write). This observation will be useful in the following section.

It remains to argue that G is acyclic if and only if $\mathcal{T}$ is robust. For this, we observe that G equals $IG_u(\mathcal{T})$.

Thus, if G is acyclic, then $IG_u(\mathcal{T})$ is acyclic, which, due to bidirectionality of $IG(\mathcal{T})$ indicates that every simple cycle in $IG(\mathcal{T})$ is a two-node cycle. By construction, these are all trivial (thus not transferable), and hence $\mathcal{T}$ must be robust.

If $\mathcal{T}$ is robust, then $IG(\mathcal{T})$ contains no transferable cycles. By construction, the latter implies that all simple cycles in $IG(\mathcal{T})$ are two-node cycles and thus that $IG_u(\mathcal{T})$ is acyclic. Hence, G is acyclic. □

4 READ UNCOMMITTED

In this section, we discuss robustness against READ UNCOMMITTED. This means that counter-example schedules can no longer take arbitrary forms but must adhere to the READ UNCOMMITTED isolation level. We therefore need additional requirements beyond non-triviality for cycles in interference graphs.

The work by Fekete et al. [15, 16] approaches the robustness problem by reasoning on cycles in interference graphs based on the types of conflicts occurring in them without taking the specific operations responsible for these conflicts into account. Types of conflicts are, for instance, write-write, write-read, and read-write dependencies between transactions. In this view, it might be tempting to think that a characterization for robustness against READ UNCOMMITTED can be found in terms of transferable cycles in $IG(\mathcal{T})$ without write-write conflicts. However, consider $\mathcal{T} = \{\mathtt{W}_1[\mathtt{x}]\mathtt{R}_1[\mathtt{y}]\mathtt{W}_1[\mathtt{z}]\mathtt{C}_1, \mathtt{W}_2[\mathtt{x}]\mathtt{R}_2[\mathtt{z}]\mathtt{W}_2[\mathtt{y}]\mathtt{C}_2\}$. Then, there is a transferable cycle $(T_1, \mathtt{R}_1[\mathtt{y}], \mathtt{W}_2[\mathtt{y}], T_2), (T_2, \mathtt{R}_2[\mathtt{z}], \mathtt{W}_1[\mathtt{z}], T_1)$ without write-write conflicts but no counter-example schedule can be found that is allowed under READ UNCOMMITTED due to the presence of the leading write to x in both T_1 and T_2. Furthermore, a cycle of a schedule allowed under READ UNCOMMITTED can still have write-write conflicts. Indeed, the schedule $s_1 = \mathtt{R}_1[\mathtt{x}]\mathtt{W}_2[\mathtt{x}]\mathtt{W}_2[\mathtt{y}]\mathtt{C}_2\mathtt{W}_1[\mathtt{y}]\mathtt{C}_1$ is allowed under READ UNCOMMITTED, since there is no dirty write but the (only) cycle in $CG(s_1)$ has a write-write conflict on y.

The higher-level explanation why it is necessary to reason about operations instead of transactions is that the isolation level READ UNCOMMITTED (and READ COMMITTED) does not guarantee atomic visibility requiring that either all or none of the updates of each transaction are visible to other transactions. More formally, a schedule s over a set of transactions $\mathcal{T}$ guarantees *atomic visibility* when $\mathtt{W}_\mathtt{i}[\mathtt{x}] <_s \mathtt{R}_\mathtt{j}[\mathtt{x}]$ iff $\mathtt{W}_\mathtt{i}[\mathtt{y}] <_s \mathtt{R}_\mathtt{j}[\mathtt{y}]$ for all $T_i, T_j \in \mathcal{T}$. For instance, the schedule $s_2 = \mathtt{R}_1[\mathtt{x}]\mathtt{R}_2[\mathtt{y}]\mathtt{W}_2[\mathtt{x}]\mathtt{W}_2[\mathtt{y}]\mathtt{C}_2\mathtt{R}_1[\mathtt{y}]\mathtt{C}_1$ is allowed under READ UNCOMMITTED but does not guarantee atomic visibility as $\mathtt{R}_1[\mathtt{x}] <_{s_2} \mathtt{W}_2[\mathtt{x}]$ but $\mathtt{W}_2[\mathtt{y}] <_{s_2} \mathtt{R}_1[\mathtt{y}]$. When an isolation level guarantees atomic visibility it suffices to reason on the level of transactions rather than on the order of operations occurring in them [11]. For READ UNCOMMITTED (and READ COMMITTED), we do need to take the ordering of operations in individual transactions into account as will become apparent in the notion of prefix-write-conflict-free cycle as defined next.

Definition 13. Let $\mathcal{T}$ be a set of transactions and let C be a cycle in $IG(\mathcal{T})$. Let $T \in \mathcal{T}$ and $b, a \in T$. Then, C is *prefix-write-conflict-free in T on operations* (b, a) if C is transferable in T on operations (b, a) and there is no write operation in $\text{prefix}_b(T)$ that conflicts with a write operation in a transaction in $C \backslash \{T\}$.[4]

Furthermore, C is *prefix-write-conflict-free* if it is prefix-write-conflict-free in T on (b, a) for some $T \in \mathcal{T}$ and some operations $b, a \in T$.

Example 14. Cycle C_1 of Example 9 is prefix-write-conflict-free in T_3 on operations $(\mathtt{W}_3[\mathtt{x}], \mathtt{W}_3[\mathtt{z}])$. Indeed, there is no write operation in T_2 or T_1 to object x. Notice that the split schedule s_1 of Example 9 is allowed under READ UNCOMMITTED. The next lemma shows that this is always the case.

LEMMA 15. *Let $\mathcal{T}$ be a set of transactions. Let C be a prefix-write-conflict-free cycle in $IG(\mathcal{T})$. Then, there is a split schedule for $\mathcal{T}$ based on C that is allowed under isolation level* READ UNCOMMITTED.

PROOF. Let $T \in \mathcal{T}$ and $b, a \in T$ such that C is prefix-write-conflict-free in T on (b, a). Let s be the split schedule based on C, T and b as defined in Definition 8. As T is the only transaction whose operations are not consecutive in s, the only possibility for a dirty write is when there is a write operation in $\text{prefix}_b(T)$ and a write operation in another transaction in C different from T that both refer to the same object. As C is prefix-write-conflict-free in T on (b, a), this cannot be the case. Therefore, s is allowed under READ UNCOMMITTED. □

We are now ready to formulate a theorem that provides a characterization for deciding robustness against READ UNCOMMITTED in terms of the existence of prefix-write-conflict-free cycles. It readily follows from Lemmas 15 and 10(3) that the existence of a prefix-write-conflict-free cycle is a sufficient condition for the existence of a counter-example schedule. The next theorem establishes that it is also a necessary condition and in addition that always a counter example in the form of a split schedule can be found.

[4]We abuse notation here and denote the set of transactions occurring in C also by C.

Theorem 16. *Let $\mathcal{T}$ be a set of transactions. The following are equivalent:*

(1) *$\mathcal{T}$ is not robust against isolation level* read uncommitted*;*
(2) *$IG(\mathcal{T})$ contains a prefix-write-conflict-free cycle; and,*
(3) *there is a split schedule s for $\mathcal{T}$ that is allowed under* read uncommitted.

Proof. *(3→1)* Immediate by Lemma 10(3).
(2→3) Follows from Lemma 15.
(1→2) Let $\mathcal{T}$ be a set of transactions that is not robust against isolation level read uncommitted. Toward a contradiction, suppose that $IG(\mathcal{T})$ contains no prefix-write-conflict-free cycle. The following is then implied by Definition 13:

(†) for every cycle C in $IG(\mathcal{T})$ that is transferable in some $T_i \in C$ and on some pair of operations (b, a), there is a write $\mathtt{W_i[x]} \in T_i$, with $\mathtt{W_i[x]} \leq_{T_i} b$, and a transaction $T_k \in C$ different from T_i with a write $\mathtt{W_k[x]} \in T_k$.

By Theorem 2 and the definition of robustness (Definition 4) there is a schedule s for $\mathcal{T}$ allowed under read uncommitted that admits a cycle C in $CG(s)$. W.l.o.g., we can assume that C is a minimal cycle, that is, there is no cycle in $CG(s)$ consisting of a strict subset of the transactions occurring in C. By Lemma 10(1), C is a transferable cycle in $IG(\mathcal{T})$. Furthermore, assumption (†) applies to C.

When C is transferable in T on some operation (b, a), we also say that T is a *breakable* transaction. The name comes from the fact that C can be split on T to create a split schedule. That is, T needs to be broken to create the split schedule.

The assumption (†) allows to derive the existence of conflicting write operations for neighboring transactions (of which at least one is breakable) in a transferable cycle. As the schedule s cannot exhibit dirty writes, the ordering of these writes in s determines the ordering of the commits of the respective transactions in s as well. The general idea is now to order neighboring transactions (w.r.t. $<_s$) for all breakable transactions and extend this partial order to a complete order for all other transactions in C. But as C is cyclic this means that there will be a transaction that is smaller than itself (w.r.t. $<_s$), which leads to the desired contradiction.

We distinguish two cases: C consists of only two edges and C contains strictly more than two edges. In the former case the simple structure allows for a more direct argument. In the latter case, we are sure that nodes have two different neighbors in the cycle but more care needs to be taken to compute the contradicting ordering in an iterative manner depending on the structure of breakable transactions.

Case 1: C contains precisely two edges. Let C be the cycle consisting of the sequence of edges $(T_1, b_1, a_2, T_2), (T_2, b_2, a_1, T_1)$. By definition of the conflict graph, we have that $b_1 <_s a_2$ and $b_2 <_s a_1$. By assumption, C is a transferable cycle in $IG(\mathcal{T})$. Then, C is either transferable in T_1 on (b_1, a_1) or C is transferable in T_2 on (b_2, a_2). We assume w.l.o.g. that C is transferable on (b_1, a_1) (otherwise, reorder the sequence of two edges in C), and, consequently, that $b_1 <_s a_1$.

From (†) it follows that there is a write $\mathtt{W_1[x]} \leq_{T_1} b_1$ in T_1 and a write $\mathtt{W_2[x]}$ in T_2. Since a_2 occurs before the commit $\mathtt{C_2}$ of T_2, the order $\mathtt{W_1[x]} \leq_{T_1} b_1 <_s a_2 <_{T_2} \mathtt{C_2}$ implies $\mathtt{W_1[x]} <_{T_1} \mathtt{C_1} <_s \mathtt{W_2[x]}$, due to the absence of dirty-writes in s (recall that s is allowed under read uncommitted).

We now consider the alternative cycle C' in $IG(\mathcal{T})$, consisting of edges $(T_1, \mathtt{W_1[x]}, \mathtt{W_2[x]}, T_2)$ and (T_2, b_2, a_1, T_1), that is transferable in T_2 on $(b_2, \mathtt{W_2[x]})$, since $b_2 <_s a_1 <_{T_1} \mathtt{C_1} <_s \mathtt{W_2[x]}$. Again due to (†), there is a write $\mathtt{W_2[y]} \leq_{T_2} b_2$ in T_2 and $\mathtt{W_1[y]}$ in T_1.

Turning back to schedule s, we observe that $\mathtt{W_2[y]} \leq_{T_2} b_2 <_s a_1 <_{T_1} \mathtt{C_1}$ implies, due the absence of dirty-writes in s, that $\mathtt{W_2[y]} <_{T_2} \mathtt{C_2} <_s \mathtt{W_1[y]}$. The latter provides the desired contradiction, as

$\mathtt{W_1[x]} <_s \mathtt{C_1} <_s \mathtt{W_2[x]} <_s \mathtt{C_2} <_s \mathtt{W_1[y]}$ cannot occur in s. Indeed, the commit is always the last operation in a transaction, thus $\mathtt{C_1} <_s \mathtt{W_1[y]}$ is not allowed.

Case 2: C **contains more than two edges.** Based on C, we construct a special cycle D in $CG(s)$ that is of the same length as C and contains precisely the same transactions as C. In addition, we construct a partial function ϵ mapping each transaction in D to one of its operations such that the following properties are true. For every edge (T_i, b_i, a_j, T_j) in D:

(i) *if ϵ is not defined for T_i but is defined for T_j then* $\epsilon(T_j) = \mathtt{C_j}$;
(ii) *if ϵ is defined for T_i and for T_j then* $\epsilon(T_i) <_s \epsilon(T_j)$; and,
(iii) *if ϵ is defined for T_i but not for T_j then $\epsilon(T_i) <_s a_j$.*

If a *total* labeling ϵ with the above conditions can be found, then we have obtained the desired contradiction. Indeed, then ϵ is defined for every transaction and it follows that $\epsilon(T_i) <_s \epsilon(T_i)$.

Next, we describe how to construct D and ϵ. Initially, we take D equal to C and ϵ as the partial mapping that labels no transaction (that is, the mapping with an empty domain). Clearly, these satisfy properties (i)–(iii).

The construction then proceeds in two phases. In the first phase, we iteratively adapt D and ϵ preserving properties (i)–(iii) and ensuring the next property:

(iv) *labeling ϵ is defined for all breakable transactions T_i in D.*

In the second phase, we show how to expand ϵ to the desired total labeling.

Phase 1: (Toward property (iv)). Before describing the procedure, we make a couple of observations. To this end, for a transaction T_i, we denote by T_{i-1} and T_{i+1} the previous and next transaction in D. That is, we assume the edges $(T_{i-1}, b_{i-1}, a_i, T_i)$ and $(T_i, b_i, a_{i+1}, T_{i+1})$.

First, when T_i is transferable on (b_i, a_i), (†) implies that some write $\mathtt{W_i[x]} \leq b_i$ in T_i conflicts with a write $\mathtt{W_k[x]}$ in one of the other transactions T_k of D. Since D is minimal and has at least three transactions, T_k is either T_{i-1} or T_{i+1}. Indeed, assume $k \neq i-1$ and $k \neq i+1$, then we can always construct a counter example to the minimality of C (recall that C and D consist of the same set of transactions). If $\mathtt{W_i[x]} <_s \mathtt{W_k[x]}$, then the sequence obtained from D by replacing

$$(T_i, b_i, a_{i+1}, T_{i+1}), \ldots, (T_{k-1}, b_{k-1}, a_k, T_k)$$

by the edge $(T_i, \mathtt{W_i[x]}, \mathtt{W_k[x]}, T_k)$ serves as a counter example. Otherwise, if $\mathtt{W_k[x]} <_s \mathtt{W_i[x]}$, then the sequence obtained from D by replacing

$$(T_k, b_k, a_{k+1}, T_{k+1}), \ldots, (T_{i-1}, b_{i-1}, a_i, T_i)$$

by the edge $(T_k, \mathtt{W_k[x]}, \mathtt{W_i[x]}, T_i)$ is a counter example to the minimality of C.

For ease of exposition, we say that a breakable transaction T_i in D is *left-breakable* if there is a write $\mathtt{W_i[x]} \leq b_i$ in T_i that conflicts with a write $\mathtt{W_{i-1}[x]}$ in T_{i-1}, and that T_i is *right-breakable* when it is not left-breakable. Note that if T_i is right-breakable, then there is a write $\mathtt{W_i[x]} \leq b_i$ in T_i that conflicts with a write $\mathtt{W_{i+1}[x]}$ in T_{i+1}.

Next, we normalize cycle D by replacing for every right-breakable transaction T_i in D the edge $(T_i, b_i, a_{i+1}, T_{i+1})$ by the edge

$$(T_i, \mathtt{W_i[x_i]}, \mathtt{W_{i+1}[x_i]}, T_{i+1}),$$

for some choice of object $\mathtt{x}_i \in \mathbf{Obj}$ with $\mathtt{W_i[x_i]} \leq b_i$ in T_i and $\mathtt{W_{i+1}[x_i]}$ in T_{i+1}. Recall that $(T_i, \mathtt{W_i[x_i]}, \mathtt{W_{i+1}[x_i]}, T_{i+1})$ is indeed an edge in CG(s), thus with $\mathtt{W_i[x_i]} <_s \mathtt{W_{i+1}[x_i]}$, because D is minimal in $CG(s)$ and with more than two transactions, thus $CG(s)$ contains (possibly multiple) edges from T_i to T_{i+1} and none from T_{i+1} to T_i.

Since D is transferable, it contains at least one breakable transaction. Therefore every transaction T_i in D is either not breakable, left-breakable, or right-breakable with $(T_i, \mathtt{W_i[x_i]}, \mathtt{W_{i+1}[x_i]}, T_{i+1})$ in D. Moreover, for at least one breakable transaction T_i in D it holds that

(‡) T_i is either left-breakable or has a non-breakable right neighbor T_{i+1}.

Indeed, if all transactions in D are right-breakable, then we have a contradiction, since the edges $(T_i, \mathtt{W_i[x_i]}, \mathtt{W_{i+1}[x_i]}, T_{i+1})$ imply $\mathtt{C_i} <_s \mathtt{C_{i+1}}$, from which it follows that $\mathtt{C_i} <_s \mathtt{C_{i+1}} <_s \cdots <_s \mathtt{C_{i-1}} <_s \mathtt{C_i}$.

To define ϵ for the breakable transactions in D, we first fix a transaction T^* in D with property (‡) and then iteratively pick the last breakable transaction occurring before and including T^* in D for which ϵ is not yet defined and define ϵ for this transaction. For each transaction T_i that we pick, we distinguish two possible cases: T_i is left-breakable and T_i is right-breakable. In the latter case, we have that some $(T_i, \mathtt{W_i[x_i]}, \mathtt{W_{i+1}[x_i]}, T_{i+1}) \in D$. Furthermore, to simplify presentation, we use ϵ' for the new ϵ.

(Case: T_i is left-breakable). This means that there is a write $\mathtt{W_i[x]} \leq_{T_i} b_i$ in T_i that conflicts with a write $\mathtt{W_{i-1}[x]}$ in T_{i-1}. Recall that D is minimal in $CG(s)$ and has more than two transactions. Therefore, $CG(s)$ contains (possibly multiple) edges from T_{i-1} to T_i and none from T_i to T_{i-1}. From this observation, we derive that $\mathtt{W_{i-1}[x]} <_s \mathtt{W_i[x]}$ and (since s is allowed under READ UNCOMMITTED) that $\mathtt{W_{i-1}[x]} <_s \mathtt{C_{i-1}} <_s \mathtt{W_i[x]}$. Let ϵ' be the labeling ϵ extended with $\epsilon'(T_{i-1}) := \mathtt{C_{i-1}}$ and $\epsilon'(T_i) := b_i$. If $\epsilon(T_{i-1})$ was already defined, then we ignore its old value.

We conclude the case by showing for the affected edges that Properties (i)–(iii) remain true for D and ϵ':

- $(T_{i-2}, b_{i-2}, a_{i-1}, T_{i-1})$: Properties (i) and (iii) follow directly from the fact that ϵ' is defined for T_{i-1} with $\epsilon'(T_{i-1}) = \mathtt{C_{i-1}}$. It remains to show Property (ii), particularly that $\epsilon'(T_{i-2}) <_s \epsilon'(T_{i-1})$ if ϵ' is defined for T_{i-2}. The latter would imply $\epsilon'(T_{i-2}) = \epsilon(T_{i-2})$. If ϵ is not defined for T_{i-1}, then $\epsilon(T_{i-2}) <_s a_{i-1} <_{T_{i-1}} \mathtt{C_{i-1}}$ (by (iii) for ϵ on T_{i-2} and T_{i-1}), and otherwise $\epsilon(T_{i-2}) <_s \epsilon(T_{i-1}) \leq_{T_{i-1}} \mathtt{C_{i-1}}$ (by (ii) for ϵ on T_{i-2} and T_{i-1}). In both cases, $\epsilon'(T_{i-2}) <_s \mathtt{C_{i-1}} = \epsilon'(T_{i-1})$.
- $(T_{i-1}, b_{i-1}, a_i, T_i)$: Properties (i) and (iii) are trivial, since ϵ' is defined for T_{i-1} and T_i. To see that Property (ii) is true, recall that $\mathtt{C_{i-1}} <_s \mathtt{W_i[x]} \leq_{T_i} b_i$.
- $(T_i, b_i, a_{i+1}, T_{i+1})$: Property (i) is trivial, since ϵ' is defined for T_i. Property (ii) is true, because if ϵ' is defined for T_{i+1}, then $\epsilon'(T_{i+1}) = \epsilon(T_{i+1}) = \mathtt{C_{i+1}}$, due to Property (i). The edge $(T_i, b_i, a_{i+1}, T_{i+1})$ then implies $\epsilon'(T_i) = b_i <_s a_{i+1} <_{T_{i+1}} \mathtt{C_{i+1}} = \epsilon'(T_{i+1})$. Property (iii) is true, because, if ϵ' is not defined for T_{i+1}, then it follows from edge $(T_i, b_i, a_{i+1}, T_{i+1})$ that $\epsilon'(T_i) = b_i <_s a_{i+1}$.

(Case: T_i is right-breakable and $(T_i, \mathtt{W_i[x_i]}, \mathtt{W_{i+1}[x_i]}, T_{i+1}) \in D$). As node labeling, we take ϵ' obtained from ϵ by setting $\epsilon'(T_i) := \mathtt{C_i}$ and $\epsilon'(T_{i+1}) := \mathtt{W_{i+1}[x_i]}$. The analysis is analogous to the previous case:

- $(T_{i-1}, b_{i-1}, a_i, T_i)$: Properties (i) and (iii) are trivial, since ϵ' is defined for T_i, with $\epsilon'(T_i) = \mathtt{C_i}$. Property (ii) is true, because if ϵ' is defined for T_{i-1}, then $\epsilon'(T_{i-1}) = \epsilon(T_{i-1}) < a_i$ due to Property (iii), and thus $\epsilon'(T_{i-1}) <_s \mathtt{C_i} = \epsilon'(T_i)$.
- $(T_i, \mathtt{W_i[x_i]}, \mathtt{W_{i+1}[x_i]}, T_{i+1})$: Properties (i) and (iii) are trivial, because ϵ' is defined for both T_i and T_{i+1}. Property (ii) follows from the observation $\mathtt{W_i[x_i]} <_s \mathtt{W_{i+1}[x_i]}$, which implies $\mathtt{W_i[x_i]} <_{T_i} \mathtt{C_i} <_s \mathtt{W_{i+1}[x_i]}$ (due to READ UNCOMMITTED). Thus, $\epsilon'(T_i) = \mathtt{C_i} <_s \mathtt{W_{i+1}[x_i]} = \epsilon'(T_{i+1})$.
- $(T_{i+1}, b_{i+1}, a_{i+2}, T_{i+2})$: Property (i) is trivial, since ϵ' is defined for T_{i+1}. To see that Property (ii) is true, assume that ϵ' is defined for T_{i+2}. Then, $\epsilon(T_{i+2}) = \epsilon'(T_{i+2})$, since $\epsilon'(T_{i+2})$ was not set

in this step. We distinguish two cases: If ϵ was defined for T_{i+1}, then $\mathtt{C}_{\mathtt{i+1}} = \epsilon(T_{i+1}) <_s \epsilon(T_{i+2})$ due to Properties (i) and (ii) on ϵ. Thus, $\epsilon'(T_{i+1}) \leq_{T_{i+1}} \mathtt{C}_{\mathtt{i+1}} <_s \epsilon(T_{i+2}) = \epsilon'(T_{i+2})$. Otherwise, if ϵ was not defined for T_{i+1}, then, due to our selection procedure, T_{i+1} cannot be breakable, implying $\epsilon'(T_{i+1}) = \mathtt{W}_{\mathtt{i+1}}[\mathtt{x}_\mathtt{i}] \leq_{T_{i+1}} b_{i+1}$, and due to edge $(T_{i+1}, b_{i+1}, a_{i+2}, T_{i+2})$ that $b_{i+1} <_s a_{i+2} <_{T_{i+2}} \mathtt{C}_{\mathtt{i+2}} = \epsilon(T_{i+2}) = \epsilon'(T_{i+2})$.

To see that Property (iii) is true we assume that ϵ' is not defined for T_{i+2}. Note that if ϵ was defined for T_{i+1}, then it was also defined for T_{i+2} (due to the order in which we pick transactions and define ϵ for them), making the case trivial. Thus, ϵ was not defined for T_{i+1}, which implies due to our selection procedure that T_{i+1} is not breakable and $\epsilon'(T_{i+1}) = \mathtt{W}_{\mathtt{i+1}}[\mathtt{x}_\mathtt{i}] \leq_{T_{i+1}} b_{i+1} <_s a_{i+2}$.

<u>Phase 2:</u> (Total ϵ). Phase 1 of the construction clearly leads to a labeling ϵ that has Properties (i)–(iv). Moreover, since the resulting cycle D has at least one transferable transaction, labeling ϵ must be defined for at least one transaction. To make ϵ total, we repeatably pick a transaction T_i such that ϵ is defined for the previous transaction T_{i-1} but not for T_i. Furthermore, let $(T_i, b_i, a_{i+1}, T_{i+1})$ be an edge in D. Then set $\epsilon'(T_i) := b_i$. Clearly, if ϵ' is defined for T_{i-1}, then $\epsilon'(T_{i-1}) <_s a_i$, due to Property (iii), and $a_i \leq_s b_i$ (because otherwise ϵ should have been defined already for T_i in the previous phase), and thus $\epsilon'(T_{i-1}) <_s \epsilon'(T_i)$. Furthermore, if $\epsilon(T_{i+1})$ is defined, then it equals $\mathtt{C}_{\mathtt{i+1}}$ (due to Property (i)). Thus, edge $(T_i, b_i, a_{i+1}, T_{i+1})$ implies $\epsilon'(T_i) = b_i <_s a_{i+1} <_{T_{i+1}} \mathtt{C}_{\mathtt{i+1}} = \epsilon(T_{i+1}) = \epsilon'(T_{i+1})$. We conclude that repeatably applying this argument indeed leads to the desired contradicting labeling ϵ, which concludes the proof. □

The following theorem establishes the complexity of deciding robustness against READ UNCOMMITTED.

THEOREM 17. ROBUSTNESS(READ UNCOMMITTED) *is* LOGSPACE-*complete.*

PROOF. The proof showing the upper bound is analogous to the proof of Theorem 12, but now we check for every pair of incident edges $\{T_i, T_j\}$ and $\{T_j, T_k\}$ in $IG(\mathcal{T})$, *and every tuple* (b_j, a_j) *of different operations* b_j *and* a_j *in* T_j *with* $b_j <_{T_j} a_j$ (thus witnessing transferability) that either T_i and T_k are the same transaction *and* none of the operations in T_i is conflicting with an operation in $\text{prefix}_{b_j}(T_j)$ *or* T_i and T_k are reachable through a path that omits T_j *and* all transactions having a write that is conflicting with a write operation in $\text{prefix}_{b_j}(T_j)$. The proof proceeds as in the proof of Theorem 12

The lower bound is by an FO-reduction from the LOGSPACE-complete undirected acyclicity problem [14] to transaction robustness against READ UNCOMMITTED. The construction of $\mathcal{T}$ for a given undirected graph G is identical to the construction presented in the proof of Theorem 12. Recall in particular that every schedule over $\mathcal{T}$ is allowed under READ UNCOMMITTED.

It remains to argue that G is acyclic if and only if $\mathcal{T}$ is robust against isolation level READ UNCOMMITTED. For this, we observe that G equals $IG_u(\mathcal{T})$.

Thus, if G is acyclic, then $IG_u(\mathcal{T})$ is acyclic, which, due to bidirectionality of $IG(\mathcal{T})$ indicates that every simple cycle in $IG(\mathcal{T})$ is a two-node cycle. By construction, these are all trivial (thus not transferable), hence $\mathcal{T}$ must be robust.

If $\mathcal{T}$ is *robust against* READ UNCOMMITTED, then $IG(\mathcal{T})$ contains no prefix-write-conflict-free cycles. Since every schedule for $\mathcal{T}$ is allowed under READ UNCOMMITTED, because every pair of conflicting operations involves precisely one write and one read (and is thus free of dirty-writes) "no prefix-write-conflict-free cycles" here implies "no cycles." By construction, the latter implies that all simple cycles in $IG(\mathcal{T})$ are two-node cycles and thus that $IG_u(\mathcal{T})$ is acyclic. Hence, G is acyclic. □

5 READ COMMITTED

Next, we discuss robustness against READ COMMITTED, which means that counter-example schedules must adhere to the READ COMMITTED isolation level. This section contains two main results: (i) a characterization of robustness against READ COMMITTED in terms of *multi-split* schedules and *multi-prefix-conflict-free* cycles (Theorem 25) and (ii) CONP-hardness of the associated decision problem (Theorem 29).

5.1 Multi-split Schedules

We start by showing that when a counter-example schedule exists, it can always take the form of a multi-split schedule based on a transferable cycle as defined below. In contrast to a split schedule where one transaction is split open and all other transactions are inserted in between in the order as they occur in the cycle, a multi-split schedule can open several transactions appearing consecutively in the cycle but needs to close them in the same order. Figure 2 provides an abstract view of a split schedule omitting the possible trailing sequence of non-interleaved transactions. To facilitate the definition of multi-split schedules, we assume that the first transaction in the cycle that the schedule is based on, is the first transaction that is opened.

Definition 18. Let $\mathcal{T}$ be a set of transactions and C a cycle in $IG(\mathcal{T})$ that is transferable in its first transaction T_1 on operations (b_1, a_1). A *multi-split schedule for $\mathcal{T}$ based on C* is any schedule of the form

$$\begin{aligned}\mathsf{prefix}_{\epsilon(T_1)}(T_1) \cdot \mathsf{prefix}_{\epsilon(T_2)}(T_2) \cdot \ldots \cdot \mathsf{prefix}_{\epsilon(T_m)}(T_m) \cdot \\ \mathsf{postfix}_{\epsilon(T_1)}(T_1) \cdot \mathsf{postfix}_{\epsilon(T_2)}(T_2) \cdot \ldots \cdot \mathsf{postfix}_{\epsilon(T_m)}(T_m) \cdot \\ T_{m+1} \cdot T_{m+2} \cdot \ldots \cdot T_n,\end{aligned}$$

with $T_1, \ldots, T_m$ denoting the transactions in C in the order as they occur and with $T_{m+1}, \ldots, T_n$ denoting the remaining transactions in $\mathcal{T}$ in an arbitrary order. Here, ϵ is a function that maps each transaction occurring in C to one of its operations and that satisfies the following conditions: for every $i > 1$,

(1) $\epsilon(T_1) = b_1$;
(2) if $\epsilon(T_{i-1}) = \mathtt{C}_{i-1}$, then $\epsilon(T_i) = \mathtt{C}_i$; and
(3) if $\epsilon(T_{i-1}) \neq \mathtt{C}_{i-1}$, then $\epsilon(T_i) = b_i$ or $\epsilon(T_i) = \mathtt{C}_i$ with the edge (T_i, b_i, a_j, T_j) in C where $j = i+1$ if $i < m$ and $j = 1$ otherwise.

The transaction T_i is called open when $\epsilon(T_i) \neq \mathtt{C}_i$ and is closed otherwise. Notice that for a closed transaction T_i, $\mathsf{prefix}_{\epsilon(T_i)}(T_i) = T_i$ and $\mathsf{postfix}_{\epsilon(T_i)}(T_i)$ is empty. A multi-split schedule is *fully split* when all transactions are open, that is, $\epsilon(T_i) \neq \mathtt{C}_i$ for all $i \in [1, m]$.

We say that s is a multi-split schedule for $\mathcal{T}$ if it is a multi-split schedule for $\mathcal{T}$ based on some cycle C. Notice that there is always a number $k > 0$ such that the first k transactions occurring in C are open and the others (if any) are closed. In a *fully split* schedule there are no closed transactions.

The next lemma establishes that a multi-split schedule gives rise to a cycle in the corresponding conflict graph.

LEMMA 19. *Let s be a multi-split schedule for a set of transactions $\mathcal{T}$ based on a cycle C in $IG(\mathcal{T})$. Then C is also a cycle in $CG(s)$.*

PROOF. Let C consist of the edges $(T_i, b_i, a_{i+1}, T_{i+1})$ for $i \in [1, n]$ with $a_{n+1} = a_1$ and $T_{n+1} = T_1$. Assume T_1 is the first transaction in C. Assume C is transferable in T_1 on (b_1, a_1). To argue that C is a cycle in $CG(s)$ as well, it suffices to show that every edge in C is an edge in $CG(s)$. To this end, consider the edge $(T_i, b_i, a_{i+1}, T_{i+1})$ in C with $i \in [1, n]$. Then, $b_i \in \mathsf{prefix}_{\epsilon(T_i)}(T_i)$. Let $i < n$. Then, since, $a_{i+1} \in T_{i+1}$ and both $\text{prefix}_{\epsilon(T_{i+1})}(T_{i+1})$ as well as $\text{postfix}_{\epsilon(T_{i+1})}(T_{i+1})$ occur after $\text{prefix}_{\epsilon(T_i)}(T_i)$ it follows that $b_i <_s a_{i+1}$. So, $(T_i, b_i, a_{i+1}, T_{i+1})$ is an edge in $CG(s)$ as well. For $i = n$, we have the edge (T_n, b_n, a_1, T_1). As C is transferable in T_1 on (b_1, a_1), it follows that $a_1 \in \text{postfix}_{b_1}(T_1)$ while $b_n \in \text{prefix}_{\epsilon(T_n)}(T_n)$. So, (T_n, b_n, a_1, T_1) is an edge in $CG(s)$ as well. □

The previous lemma does not imply that s is allowed under READ COMMITTED. To this end, we introduce the definition of a multi-prefix-conflict-free cycle. First, we define the following notions. Let $\mathcal{T}$ be a set of transactions, C a cycle in the interference graph $IG(\mathcal{T})$, and T a transaction in $\mathcal{T}$. Then there is precisely one edge of the form (T, b, a, T') in C for some $b \in T$, $T' \in \mathcal{T}$, and $a \in T'$. For ease of notation, we write $b_C(T)$ to denote b and $a_C(T)$ to denote a. When C is clear from the context, we also write $a(T)$ and $b(T)$ for $a_C(T)$ and $b_C(T)$, respectively.

In the following definition, T and T' intuitively refer to the first open and last open transaction in the multi-split schedule that can be constructed from a multi-prefix-conflict-free cycle.

Definition 20. Let $\mathcal{T}$ be a set of transactions and let C be a cycle in $IG(\mathcal{T})$ containing transactions T and T'. Then C is *multi-prefix-conflict-free* in T and T' if C is transferable in T and for every transaction T_i that is equal to T' or occurs before T' in $C[T]$ there is no write operation in $\text{prefix}_{b(T_i)}(T_i)$ that

- conflicts with a read or write operation in $\text{prefix}_{b(T_j)}(T_j)$ of some transaction T_j occurring after T_i but before or equal to T' in $C[T]$; or
- conflicts with a read or write operation in some transaction T_j occurring after T' in $C[T]$; or
- conflicts with a read or write operation in $\text{postfix}_{b(T_j)}(T_j)$ of some transaction T_j occurring strictly before T_i in $C[T]$.

The next lemma says that when a multi-prefix-conflict-free cycle can be found, a corresponding counter-example multi-split schedule witnessing non-robustness against READ COMMITTED can be constructed. In Theorem 25, we show that the latter is also a necessary condition.

LEMMA 21. *Let $\mathcal{T}$ be a set of transactions. Let C be a cycle in $IG(\mathcal{T})$ that is multi-prefix-conflict-free in T and T'. Then, there is a multi-split schedule for $\mathcal{T}$ based on C that is allowed under isolation level* READ COMMITTED.

PROOF. Let C be a multi-prefix-conflict-free cycle in $IG(\mathcal{T})$ in T and T'. Let C consist of the edges $(T_i, b_i, a_{i+1}, T_{i+1})$ for $i \in [1, n]$ with $a_{n+1} = a_1$ and $T_{n+1} = T_1$. Assume $T = T_1$ is the first transaction in C; otherwise, take C as $C[T]$. Assume C is transferable in T on (b_1, a_1). Let s be the multi-split schedule based on C where ϵ is defined as follows: Let $\epsilon(T_i) = b_i$ for every transaction equal to T' or occurring before T' in C. Furthermore, define $\epsilon(T_i) = \mathtt{C_i}$ for every other transaction in C. The trailing transactions in s that are not in C are arbitrarily ordered.

It remains to argue that s is allowed under READ COMMITTED. For this, let a be a write operation in some transaction T_i and b a read or write operation in some transaction T_j with $a <_s b$. We next argue that it always follows that $a <_s \mathtt{C_i} <_s b$. So, no dirty read or write occurs in s. If T_j is not in C, then it follows from the construction of s that $a <_s \mathtt{C_i} <_s b$. Furthermore, if T_i is not in C, then it follows that T_j is also not in C and the previous case applies. Therefore, assume T_i and T_j are in C. We distinguish two cases:

- Assume T_i occurs before T_j in C.
  - Let $a \in \text{prefix}_{\epsilon(T_i)}(T_i)$. It follows from the second condition in Definition 20 that T_j occurs before T' or is equal to T'. Then the first condition in Definition 20 prohibits that $b \in \text{prefix}_{\epsilon(T_j)}(T_j)$. So, $b \in \text{postfix}_{\epsilon(T_j)}(T_j)$ and T_j must be open. But as T_i occurs before T_j in C this means that by construction $\mathtt{C_i} <_s b$.

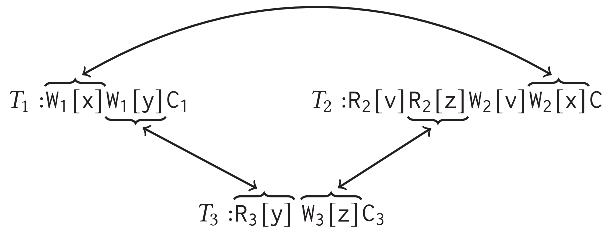


Fig. 4. $IG(\mathcal{T})$ for $\mathcal{T} = \{T_1, T_2, T_3\}$ as defined in Example 22.

  – Let $a \in \text{postfix}_{\epsilon(T_i)}(T_i)$ and T_i be open. Then $b \in \text{postfix}_{\epsilon(T_j)}(T_j)$ and T_j is open and by construction $\mathtt{C}_i <_s b$.

- We next argue that T_j cannot occur before T_i in C. Toward a contradiction, assume T_j occurs before T_i in C. It cannot be the case that $a \in \text{prefix}_{\epsilon(T_i)}(T_i)$. Indeed, as $a <_s b$, it follows that $b \in \text{postfix}_{\epsilon(T_j)}(T_j)$ but this cannot be the case due to the third condition of Definition 20. It can also not be the case that $a \in \text{postfix}_{\epsilon(T_i)}(T_i)$ as this would imply that $b <_s a$. Therefore, T_j cannot occur before T_i in C.

We conclude that s is indeed allowed under READ COMMITTED. □

Example 22. Consider $\mathcal{T} = \{T_1, T_2, T_3\}$ with $T_1 = \mathtt{W}_1[\mathtt{x}]\mathtt{W}_1[\mathtt{y}]\mathtt{C}_1$, $T_2 = \mathtt{R}_2[\mathtt{v}]\mathtt{R}_2[\mathtt{z}]\mathtt{W}_2[\mathtt{v}]\mathtt{W}_2[\mathtt{x}]\mathtt{C}_2$ and $T_3 = \mathtt{R}_3[\mathtt{y}]\mathtt{W}_3[\mathtt{z}]\mathtt{C}_3$. Then $IG(\mathcal{T})$ is depicted in in Figure 4. The cycle C consisting of the following edges

$$(T_1, \mathtt{W}_1[\mathtt{x}], \mathtt{W}_2[\mathtt{x}], T_2), (T_2, \mathtt{R}_2[\mathtt{z}], \mathtt{W}_3[\mathtt{z}], T_3), (T_3, \mathtt{R}_3[\mathtt{y}], \mathtt{W}_1[\mathtt{y}], T_1)$$

is multi-prefix-conflict-free in T_1 and T_2. The multi-split schedule s for $\mathcal{T}$ based on C where T_1 and T_2 are open and T_3 is closed is as follows:

$$\underbrace{\mathtt{W}_1[\mathtt{x}]}_{\text{prefix}_{b_1}(T_1)}\ \underbrace{\mathtt{R}_2[\mathtt{v}]\mathtt{R}_2[\mathtt{z}]}_{\text{prefix}_{b_2}(T_2)}\ \underbrace{\mathtt{R}_3[\mathtt{y}]\mathtt{W}_3[\mathtt{z}]\mathtt{C}_3}_{T_3}\ \underbrace{\mathtt{W}_1[\mathtt{y}]\mathtt{C}_1}_{\text{postfix}_{b_1}(T_1)}\ \underbrace{\mathtt{W}_2[\mathtt{v}]\mathtt{W}_2[\mathtt{x}]\mathtt{C}_2}_{\text{postfix}_{b_2}(T_2)},$$

with $b_1 = \mathtt{W}_1[\mathtt{x}]$ and $b_2 = \mathtt{R}_2[\mathtt{z}]$. Notice that s is allowed under READ COMMITTED.

In the proof of Theorem 25, we show that any counter-example schedule witnessing non-robustness against READ COMMITTED can be transformed into one that is a multi-split schedule. Basically, in a multi-split schedule every transaction is represented by one or two blocks of consecutive operations. Indeed, an open transaction is represented by two blocks while closed transactions as well as trailing transactions are represented by one block. We refer to such blocks of consecutive operations within a transaction as a *chunk*. Formally, in a schedule s for $\mathcal{T}$, we call a maximal sequence of consecutive operations from the same transaction T a *chunk of* T in s. For instance, in Figure 1, T_1 is represented in s_1 by one chunk ($\mathtt{W}_1[\mathtt{x}]\mathtt{R}_1[\mathtt{z}]\mathtt{W}_1[\mathtt{y}]\mathtt{C}_1$) while T_2 is represented by two chunks ($\mathtt{W}_2[\mathtt{z}]$ and $\mathtt{R}_2[\mathtt{y}]\mathtt{W}_2[\mathtt{x}]\mathtt{C}_2$).

Let T be a transaction. A subsequence B of T is a sequence of consecutive operations in T. If a is the next operation in T following the last operation in B, then $B \cdot a$ is the subsequence B extended with a. Let $\mathcal{T}$ be a set of transactions and s be a schedule for $\mathcal{T}$. Let $T \in \mathcal{T}$, and let $B \cdot a$ be a subsequence of T. Then we denote by $s(B; a)$ the schedule obtained from s by first removing all operations in B in s and then inserting them just before a in s. More formally, let $s = s_1 \cdot a \cdot s_2$. Then, $s(B; a)$ is the schedule $s_1' \cdot B \cdot a \cdot s_2$ where s_1' is obtained from s_1 by deleting every operation in B. Such actions will be performed to merge chunks in a schedule in the proof of the following theorem.

LEMMA 23. *Let $\mathcal{T}$ be a set of transactions and s a schedule for $\mathcal{T}$ allowed under isolation level $\mathcal{I} \in$ {NO ISOLATION, READ UNCOMMITTED, READ COMMITTED}. Let $B \cdot a$ be a subsequence of some* *transaction $T_i \in \mathcal{T}$. The schedule $s(B; a)$ for $\mathcal{T}$ is allowed under $\mathcal{I}$ if at least one of the following conditions is true:*

(1) *For every operation c that conflicts with an operation d in B, we have $c <_s d$ or $\mathtt{C_k} <_s a$, with $\mathtt{C_k}$ the commit of the transaction that c is in.*
(2) *Operation a equals $\mathtt{C_i}$ and T_i is the transaction whose commit occurs last in s.*
(3) *For every operation c that conflicts with an operation d in B we have $c <_s d$ or $a <_s c$.*

Proof. Observe that Condition (2) implies Condition (1), since $\mathtt{C_k} <_s \mathtt{C_i} = a$ follows from the assumption that T_i is the transaction whose commit occurs last in s. In the remainder of the proof, we show Condition (1) and Condition (3). Let $s' = s(B; a)$.

(1) For this, let $c_h \in T_h$ and $d_j \in T_j$ be two arbitrary conflicting operations with $c_h <_{s'} d_j$. Toward a contradiction, suppose that c_h and d_j witness a forbidden phenomenon in s' for isolation level $\mathcal{I}$ (i.e., $c_h <_{s'} d_j <_{s'} \mathtt{C_h}$), that is, a dirty-write if $\mathcal{I} =$ READ UNCOMMITTED and a dirty-write or dirty-read if $\mathcal{I} =$ READ COMMITTED. The proof is by case distinction:

- If $c_h \notin B$ and $d_j \notin B$, then the proof is straightforward. Indeed, the relative order among c_h, d_j, and $\mathtt{C_h}$ is identical in s and s'. Therefore, either c_h and d_j do not witness a forbidden phenomenon in s' or the phenomenon is already present in s. Both contradict with our assumptions.
- If $c_h \in B$, then $T_h = T_i$ and $c_h <_{T_h} a$. By Condition (1), $d_j <_s c_h$ or $\mathtt{C_j} <_s a$. Note that, since s' is constructed from s by moving operations in B to the right, $c_h <_{s'} d_j$ implies $c_h <_s d_j$. We conclude that $d_j <_s \mathtt{C_j} <_s a$, and hence $d_j <_{s'} \mathtt{C_j} <_{s'} c_h$, contradicting our assumption that $c_h <_{s'} d_j$.
- If $d_j \in B$, then $T_j = T_i$ and $d_j <_{T_j} a$. By Condition (1), $c_h <_s d_j$ or $\mathtt{C_h} <_s a$. Note that, since s' is constructed from s by moving operations in B to the right, $d_j <_{s'} \mathtt{C_h}$ implies $d_j <_s \mathtt{C_h}$. If $c_h <_s d_j$, then the relative order among c_h, d_j and $\mathtt{C_h}$ is identical in s and s', again leading to a contradiction. We conclude that $d_j <_s \mathtt{C_h} <_s a$. But then $c_h <_{s'} \mathtt{C_h} <_{s'} d_j$, contradicting our assumption that c_h and d_j witness a forbidden phenomenon.

We conclude that s' is indeed allowed under $\mathcal{I}$.

(3) The proof is analogous to the proof for Condition (1). Let c_h and d_j be again two arbitrary conflicting operations with $c_h <_{s'} d_j$ that we assume to witness a forbidden phenomenon for isolation level $\mathcal{I}$. If $c_h \notin B$ and $d_j \notin B$, then the proof argument is the same as in the proof for Property (1). The other two cases are as follows:

- If $c_h \in B$, then $T_h = T_i$ and $c_h <_{T_h} a$. By Condition (3), $d_j <_s c_h$ or $a <_s d_j$. Analogous to the proof for Condition (1), the former cannot happen, and hence $c_h <_s a <_s d_j$, implying that the relative order among c_h, d_j, and $\mathtt{C_h}$ is identical in s and s', again leading to a contradiction.
- If $d_j \in B$, then $T_j = T_i$ and $d_j <_{T_j} a$. By Condition (3), $c_h <_s d_j$ or $a <_s c_h$. The former case is analogous to the proof for Condition (1), implying that the relative order among c_h, d_j, and $\mathtt{C_h}$ is identical in s and s'. The latter case cannot occur, as $d_j <_s a <_s c_h$ implies $d_j <_{s'} c_h$ by construction of s' from s, contradicting our assumption that $c_h <_{s'} d_j$. □

The following lemma deals with the case where $\mathcal{T}$ contains precisely two transactions.

Lemma 24. *Let $\mathcal{T}$ be a set containing precisely two transactions. If $\mathcal{T}$ is not robust against isolation level* READ COMMITTED, *then there is a multi-split schedule s for $\mathcal{T}$ that is allowed under* READ COMMITTED.

Proof. Let s be a schedule for $\mathcal{T}$ that is allowed under READ COMMITTED and contains a cycle. We call the transaction whose commit occurs first in s transaction T_1, and the other transaction

T_2. Let c be the first operation from T_2 that conflicts with an operation d from T_1 such that $c <_s d$. (Notice that c and d exist, due to existence of a cycle C in $CG(s)$.) Next, we distinguish two cases as follows:

(Case: There is an operation a from T_1 that occurs before c in s and conflicts with an operation b from T_2). Let a be the last such operation in s. Let s' be the schedule obtained from s by moving all operations from T_2 occurring after c to the chunk with $\mathtt{C}_2$; all operations from T_2 occurring before c to the chunk with c; all operations from T_1 occurring after a and before c to the chunk with $\mathtt{C}_1$.

That s' is allowed under READ COMMITTED is straightforward by application of Lemma 23 on the three steps of the construction. Indeed, due to $\mathtt{C}_1 <_s \mathtt{C}_2$, the first step of the construction satisfies Condition (2); since c is the first operation from T_2 in s that conflicts with an operation on its right, the second step satisfies Condition (1); by choice of a, the operations between a (inclusive) and c (exclusive) are not conflicting with operations from T_2 and are inserted right before the first operation of T_1 that occurs after c, hence Condition (1) applies.

We conclude the case by observing that s' is indeed a multi-split schedule for $\mathcal{T}$ based on cycle $(T_1, a, b, T_2), (T_2, c, d, T_1)$, and function ϵ with $\epsilon(T_1) := a$ and $\epsilon(T_2) := c$.

(Case: Otherwise). We assume that none of the operations from T_1 occurring before c in s conflicts with an operation from T_2. Let s' be the schedule obtained from s by moving all operations from T_2 occurring after c to the chunk with $\mathtt{C}_2$; all operations from T_1 to the chunk with $\mathtt{C}_1$.

To see that s' is allowed under READ COMMITTED, we make the following observations: The first step of the construction satisfies Lemma 23(2), since T_2 commits last in s; the second step of the construction satisfies Lemma 23(1) by the assumption of the case.

Recall that there is an edge (T_1, a, b, T_2) in C for some operations a from T_1 and b from T_2 with $a <_s b$. By assumption of the case, we have $c <_s a$ and thus $a <_{s'} b$ (by construction of s').

Now it is straightforward to see that s' is a multi-split schedule for $\mathcal{T}$ based on the cycle $(T_2, c, d, T_1), (T_1, a, b, T_2)$, and function ϵ with $\epsilon(T_2) := c$ and $\epsilon(T_1) := \mathtt{C}_1$. □

We are now ready to prove the main theorem of this section.

THEOREM 25. *Let $\mathcal{T}$ be a set of transactions. The following are equivalent:*

(1) *$\mathcal{T}$ is not robust against isolation level* READ COMMITTED*;*
(2) *$IG(\mathcal{T})$ contains a multi-prefix-conflict-free cycle; and*
(3) *there is a multi-split schedule s for $\mathcal{T}$ that is allowed under* READ COMMITTED.

PROOF. *(3) → (2)* Let s be the assumed multi-split schedule for $\mathcal{T}$ based on a cycle C that is allowed under READ COMMITTED. Then, C is in $CG(s)$ by Lemma 19. Let $T \in C$ be the first transaction that appears in s. Let T' denote the last transaction in C that appears with two chunks in s. Then, C is multi-prefix-conflict-free in T and T'. Indeed, every transaction T_i equal to T' or occurring before T' in C has exactly two chunks in s. Assume there is a write operation a in $\mathrm{prefix}_{b_i}(T_i)$ (with $(T_i, b_i, a_{i+1}, T_{i+1})$ in C) and a conflicting read or write operation b in $\mathrm{prefix}_{b_j}(T_j)$ for transaction T_j occurring after T_i in C (with $(T_j, b_j, a_{j+1}, T_{j+1})$ in C). Then, we have by definition of multi-split schedule that $a <_s b <_s \mathtt{C}_\mathtt{i}$, which contradicts with s being allowed under READ COMMITTED. The case b in $\mathrm{postfix}_{b_j}(T_j)$ with T_j occurring before T_i in C implies $a <_s b <_s \mathtt{C}_\mathtt{i}$ as well.

(2) → (1) Follows immediately, as by Lemmas 21 and 19 there is a schedule s for $\mathcal{T}$ that is allowed under READ COMMITTED and that has a cycle in $CG(s)$.

(1) → (3) By Theorem 2 there is a schedule s_0 for $\mathcal{T}$ allowed under READ COMMITTED with a cycle C in its conflict graph.

Let $\mathcal{U} \subseteq \mathcal{T}$ denote the transactions occurring in C and let s be the schedule obtained from s_0 by removing all operations from transactions not occurring in C. Notice that C is a cycle in the conflict graph of s and that s is a schedule for $\mathcal{U}$ allowed under READ COMMITTED. Moreover, if a multi-split schedule s' exists for $\mathcal{U}$ that is allowed under READ COMMITTED, then we can easily obtain a multi-split schedule for $\mathcal{T}$ allowed under READ COMMITTED by appending to s' all missing transactions (those in $\mathcal{T} \setminus \mathcal{U}$) in a serial fashion.

The case where $\mathcal{U}$ contains precisely two transactions is treated in Lemma 24. Henceforth, we assume that $\mathcal{U}$ contains at least three transactions. Moreover, we assume that the following property applies to s and C:

(i) C is minimal in $CG(s)$ and contains at least three transactions; no schedule for $\mathcal{U}$ allowed under READ COMMITTED exists with a cycle in its conflict graph mentioning a strict subset of the transactions in C. Furthermore, s is allowed under READ COMMITTED.

If Property (i) holds for a schedule s, then the following property is immediate:

(†) Let c and d be two conflicting operations from two different transactions T_i and T_j in C. $c <_s d$ iff T_j is the transaction occurring directly before T_i in C, and $d <_s c$ iff T_i is the transaction occurring directly before T_j in C.

Indeed, any other conflict would introduce an additional edge between two transactions in C, thereby contradicting the assumption that C is a minimal cycle in $CG(s)$.

The construction requires four phases. In each phase, we transform schedule s one step closer to the desired form. Eventually, we obtain a schedule s' for $\mathcal{U}$ satisfying Properties (i)–(v):

(ii) Every transaction T_i consists either of only one chunk or exactly two chunks in s'. In the latter case, the last operation of the first chunk of T_i conflicts with an operation from transaction T_{i+1} occurring after T_i in C.

(iii) In the following, let T_1 be the transaction whose first operation occurs first in s'. Then T_1 consists of two chunks in s'. Furthermore, all pairs of chunks in s' between the first and last chunk of T_1 and all pairs of chunks in s' after the last chunk of T_1 appear in the same order as their corresponding transactions appear in $C[T_1]$. That is, for each such pair of chunks B_i and B_j belonging to respectively transactions T_i and T_j, chunk B_i occurs before B_j in s' iff T_i is situated before T_j in $C[T_1]$.

(iv) Every transaction (except T_1) has a chunk between the first and last chunk of T_1.

(v) If T_i consists of only one chunk, then the transaction T_{i+1} occurring after T_i in C (unless it is T_1) consists of only one chunk.

Notice that a schedule s and cycle C having Properties (i)–(v) indeed represent a multi-split schedule based on C that is allowed under READ COMMITTED, with as ϵ the mapping that maps T_i on the last operation of its first chunk in s, which is either some read or write operation from T_i (if T_i has two chunks) or $\mathtt{C}_i$ (if T_i has only one chunk).

Each of the four phases is detailed below. For convenience of notation, we refer in each phase by s' to the new version of s.

Phase 1: *From a schedule s for $\mathcal{U}$ allowed under* READ COMMITTED *with a cycle C in its conflict graph and with Property (i) we construct a schedule s' for $\mathcal{U}$ allowed under* READ COMMITTED *with cycle $C' \in CG(s)$ and Properties (i) and (ii).* For the construction, we iterate over the transactions in $\mathcal{U}$ in the opposite order as defined by C, starting from the transaction whose commit occurs last in s. For each visited transaction, we verify that it does not contradict Property (ii). If it does, then we rewrite s to a new schedule s' in which the property is made true for T_i and remains true for all earlier visited transactions. We continue the iterative process on the new schedule s' until Property (ii) is true.

The above procedure terminates, as we never split chunks from other transactions than the selected one. Hence, the only possible side effect on a transaction with Property (ii) in s is that its two chunks may become a single chunk in s'.

Notice that our picking order has the following implications: The first transaction T_i that we pick has property $\mathtt{C_{i+1}} <_s \mathtt{C_i}$, with T_{i+1} the transaction following T_i in C. Indeed, we start with the transaction that commits last in s. For every next transaction T_i, we can assume that Property (ii) is already true for T_{i+1}.

For the rewriting step, we distinguish three cases:

(Case: $\mathtt{C_{i+1}} <_s \mathtt{C_i}$). Let b be the first operation of T_i in s that conflicts with an operation from T_{i+1}. Then let s' be the schedule obtained by (I) removing in s all operations in $\text{prefix}_b(T_i)$ except b and inserting them in front of b and (II) removing all operations in $\text{postfix}_b(T_i)$ except $\mathtt{C_i}$ and inserting them in front of $\mathtt{C_i}$.

The resulting schedule s' is allowed under READ COMMITTED, because both steps (I) and (II) satisfy the assumptions of Lemma 23(1). Indeed, for (I) it follows from the choice of b that all operations c conflicting with an operation d in $\text{prefix}_b(T_i)$ (except operation b) are from transactions different from T_{i+1}. Due to Property (i) and (†) these operations c are from transaction T_{i-1} and thus occur before d in s. For (II), let c be an operation conflicting with an operation d in $\text{postfix}_b(T_i)$. Toward Lemma 23(1), we need to show that $c <_s d$ or $\mathtt{C_k} <_s \mathtt{C_i}$, with $\mathtt{C_k}$ the commit of the transaction that c is in. If $c <_s d$, then the argument is immediate. Otherwise, if $d <_s c$, then by Property (i) and (†) on s we conclude that c is an operation in T_{i+1}. The desired $\mathtt{C_{i+1}} <_s \mathtt{C_i}$ is now immediate from the condition of the case.

Replacing the edge between T_i and T_{i+1} in C by (T_i, b, c, T_{i+1}), with c an operation from T_{i+1} that b conflicts with, results in a cycle that is in $CG(s')$. Since C' mentions the same transactions as C, Property (i) straightforwardly transfers from s and C to s' and C'. Notice also that b (which is conflicting by assumption) is the last operation of the first chunk of T_i in s', thus s' has Property (ii) for transaction T_i.

(Case: $\mathtt{C_i} <_s \mathtt{C_{i+1}}$ and there is an operation b in T_i that conflicts with an operation e from T_{i+1} with $b <_s e <_s \mathtt{C_i}$). Let b denote the last operation in s with this property.

Let s' be the schedule obtained by (I) removing in s all operations from $\text{prefix}_b(T_i)$ except b and inserting them in front of b and (II) removing all operations in $\text{postfix}_b(T_i)$ except $\mathtt{C_i}$ and inserting them in front of $\mathtt{C_i}$.

To see that s' is allowed under READ COMMITTED, we argue that both steps (I) and (II) satisfy the assumptions of Lemma 23(3). For step (I), this follows from the observation that T_{i+1} already has Property (ii) due to the order in which we select transactions. Existence of b thus implies that the first chunk of T_{i+1} is located between b and $\mathtt{C_i}$ in s. From this, we infer that for every operation c that conflicts with an operation d in $\text{prefix}_b(T_i)$, we either have that $c <_s d$ or, if $d <_s c$, that c is from T_{i+1}, due to Property (i) and (†) on s and thus that $b <_s c$. For step (II), Lemma 23(3) applies if for every operation c that conflicts with an operation d in $\text{postfix}_b(T_i)$ we have $c <_s d$ or $\mathtt{C_i} <_s c$. To this end, if $d <_s c$, then c is an operation in T_{i+1} by Property (i) and (†) on s. The desired $\mathtt{C_i} <_s c$ then follows from our choice of b. Indeed, since $b <_{T_i} d$, the property $d <_s c <_s \mathtt{C_i}$ would contradict our choice of b.

Due to the above observations and the fact that b is the last operation of the first chunk of T_i in s', Property (ii) is indeed true for transaction T_i in s'.

Notice that the above analysis implies that cycle C remains a cycle in $CG(s')$. Hence, let C' equal C. Now it follows straightforwardly from Property (i) on s and C that Property (i) is true for s' and C'.

(Case: $\mathtt{C_i} <_s \mathtt{C_{i+1}}$ and there is no operation b in T_i that conflicts with an operation e from T_{i+1} with $b <_s e <_s \mathtt{C_i}$). Let s' be the schedule obtained by removing all operations from T_i except $\mathtt{C_i}$ from s and inserting them in front of $\mathtt{C_i}$.

To see that s' is allowed under READ COMMITTED, we apply Lemma 23(3) by showing that for every operation c that conflicts with an operation d in T_i we have $c <_s d$ or $\mathtt{C_i} <_s c$. More precisely, we show that if $d <_s c$, then $\mathtt{C_i} <_s c$. Since $d <_s c$, we derive from Property (i) and (†) that c is an operation in T_{i+1}. Then, $\mathtt{C_i} <_s c$ follows from the assumption on this case. Indeed, $d <_s c <_s \mathtt{C_i}$ contradicts our assumption that no such operation d in T_i exists.

We conclude that Property (ii) is indeed true for T_i in s', since T_i now has only one chunk.

Here, again, we let C' equal C, as it is indeed a cycle in $CG(s')$ (inferred from the earlier analysis on s'). That Property (i) is true for s' and C' follows immediately from Property (i) on s, the fact that s' is allowed under READ COMMITTED and because C' mentions the same transactions as C.

Phase 2: *From a schedule s for $\mathcal{U}$ allowed under* READ COMMITTED *with a cycle C in its conflict graph and with Properties (i) and (ii) we construct a schedule s' for $\mathcal{U}$ allowed under* READ COMMITTED *with cycle $C' \in CG(s)$ and Properties (i)–(iii).*

In the following, let T_1 be the transaction whose first operation occurs first in s. By Property (ii), T_1 consists of two chunks in s. Indeed, if T_1 would consist of only one chunk, then by our choice of T_1 every operation in T_n (the operation immediately before T_1 in C) would occur after $\mathtt{C_1}$ in s, thereby contradicting the edge from T_n to T_1 in C.

Let s' be the schedule obtained by sorting in s all chunks between the first chunk of T_1 and last chunk of T_1 based on the order of the transaction that they are part of in $C[T_1]$ and by sorting all chunks occurring after $\mathtt{C_1}$ according to the same order. Let C' equal C.

That s' is allowed under READ COMMITTED follows straightforwardly from the following observation: Due to Property (i) and (†), an operation in a chunk from some transaction T_i can only conflict with an operation in chunks from transactions T_{i-1} and T_{i+1}. Due to minimality of C in $CG(s)$ and the fact that $\mathcal{U}$ (thus also C) has three or more transactions, it follows that for chunks from transactions T_i and T_{i+1}, either they are already in the correct order, or they contain no conflicting operations and thus can be swapped safely. Since we do not swap chunks containing conflicts, cycle C' is indeed a cycle in $CG(s')$.

Property (i) on s' and C' follows from the fact that Property (i) is true on s and C and because C' equals C. Property (ii) follows from the fact that Property (ii) is true on s and because we do not split chunks to obtain s'.

Phase 3: *From a schedule s for $\mathcal{U}$ allowed under* READ COMMITTED *with a cycle C in its conflict graph and with Properties (i)–(iii) we construct a schedule s' for $\mathcal{U}$ allowed under* READ COMMITTED *with cycle $C' \in CG(s)$ and Properties (i)–(iv).*

Recall that we have chosen T_1 such that its first operation is the first operation in s. Let T_i be the last transaction (w.r.t. the order defined in $C[T_1]$) without chunk between the first and last chunk of T_1 in s. Notice that $i < n$, because $i = n$ would imply that every operation in T_n (the transaction immediately preceding T_1 in C) occurs after $\mathtt{C_1}$ in s, thereby contradicting the edge from T_n to T_1 in C. Furthermore, transaction T_{i+1} must have two chunks in s: one before $\mathtt{C_1}$ and one after $\mathtt{C_1}$. The former is immediate by our choice of T_i, whereas the latter follows from the observation that T_{i+1} must have a chunk that occurs after one of the chunks of T_i (which all occur after $\mathtt{C_1}$) to witness the edge from T_i to T_{i+1} in C. We will denote the last operation occurring in the first chunk of T_{i+1} by a.

Let s' be the schedule obtained by moving all chunks occurring before the first chunk of T_{i+1} in s to their corresponding chunk after a (thereby closing these transactions) or inserting on the right place after $\mathtt{C_1}$ w.r.t. the order defined by $C[T_1]$ (if the transaction has only one chunk in s). Let C' equal C. We emphasize in particular that T_{i+1} is now the transaction whose first operation is first in the constructed s'.

That schedule s' is allowed under READ COMMITTED follows from Lemma 23; particularly the fact that Lemma 23(3) applies to each individual swap. Property (i) follows from the assumption that Property (i) is true on s and C and by construction of C' (which equals C). Property (ii) follows from the assumption that Property (ii) is true on s and because we do not split chunks to obtain s'. Property (iii) and (iv) follow directly from the construction, taking T_{i+1} as T_1. Indeed, we do not split chunks and all repositionings are w.r.t. the order of transactions in C. By choice of T_i all transactions occurring between T_{i+1} and T_1 in C already had a chunk between the first chunk of T_{i+1} and the last chunk of T_1 (and possibly a second chunk occurring after the second chunk of T_{i+1}). Transactions T_1 until T_i, however, are always closed in s', and their only chunk appears between the first and last chunk of T_{i+1} in the order specified by $C[T_1]$. Indeed, each of these transactions either already appeared closed before the first chunk of T_{i+1} in the schedule s, or they are closed by the construction of s'. In the former case, they are moved to their correct position (relative to $C[T_1]$) after the second chunk of T_1 but before the second chunk of T_{i+1}. In the latter case, the chunk representing the first part of the transaction is moved toward the chunk representing the second part, which by Property (iii) on s was already positioned on the correct position (relative to $C[T_1]$) after the second chunk of T_1, but before the second chunk of T_{i+1}.

Phase 4: *From a schedule s for $\mathcal{U}$ allowed under* READ COMMITTED *with a cycle C in its conflict graph and with Properties (i)–(iv) we construct a schedule s' for $\mathcal{U}$ allowed under* READ COMMITTED *with cycle $C' \in CG(s)$ and Properties (i)–(v).*

Let s' be the schedule obtained from s by iteratively picking a transaction T_i having two chunks in s, with $i \neq 1$, and with T_{i-1} having only one chunk, and then removing the second chunk of T_i and inserting it immediately after its first chunk.

This procedure clearly leads to a schedule with Property (v). The resulting schedule s' is also allowed under READ COMMITTED. Indeed, suppose toward a contradiction that a pair of conflicting operations c and d exist witnessing a forbidden phenomenon for READ COMMITTED. Then either c or d must be from T_i (as otherwise the phenomenon already occurred in s). Without loss of generality, we furthermore assume that $c <_s d$ (notice that we can always swap c and d if this is not the case). If c is from T_i, then it follows from the construction that $c <_{s'} \mathtt{C_i} <_{s'} d$, which contradicts with our assumption that c and d witness a forbidden phenomenon. Similarly, if d is from T_i, then c must be from T_{i-1} (due to Property (i) on s and C). By our choice of T_i, the transaction T_{i-1} has only one chunk in s (and therefore in s' as well), which implies $c <_{s'} \mathtt{C_{i-1}} <_{s'} d$.

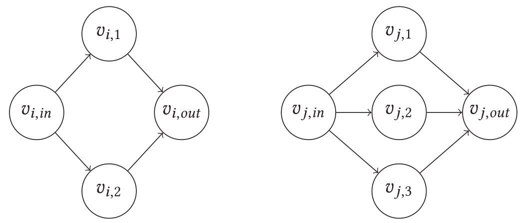
Properties (ii)–(iv) transfer from s to s', because we do not split chunks and because we do not remove chunks located between the first and second chunk of T_1. □

5.2 Intermezzo: Properly Colored Cycles

In this section, we study the complexity of a decision problem over colored graphs. Even though the problem is not directly related to deciding robustness, the reduction we present provides the no-frills intuition that will be central in the more complex reduction presented next in Section 5.3.

A *(vertex-)colored graph* is a tuple $G = (V, E, K, f)$ where V is a finite set of nodes, $E \subseteq V \times V$ is the set of edges, K is a finite set of colors, and f maps each vertex in V to a color in K. As before, a cycle C is a non-empty sequence of edges $(v_1, v_2), (v_2, v_3), \ldots, (v_n, v_1)$ such that every vertex in V does not occur in C or occurs precisely twice. The latter in particular means that C is simple. We say that C is *properly colored* if for each two vertices v_1 and v_2 occurring in C (not necessarily adjacent in C), $(v_1, v_2) \in E$ implies $f(v_1) \neq f(v_2)$. So, the induced subgraph of G determined by the vertices occurring in C should color adjacent vertices differently.

Let PROPERLYCOLOREDCYCLE be the problem to decide if a given colored graph contains a properly colored cycle. In this section, we show the following result:



(a) Variable gadget $G(x_i)$. (b) Clause gadget $G(C_j)$.

Fig. 5. Gadgets for the construction of $G(\varphi)$.

Proposition 26. ProperlyColoredCycle *is* np-*complete.*

As the upper bound is straightforward, it remains to argue that ProperlyColoredCycle is also np-hard. The proof is by a reduction from 3SAT. To this end, let φ be a propositional logic formula in 3CNF, and let $Vars(\varphi)$ be the set of variables occurring in φ. We recall that φ is a conjunction of clauses C_j of the form $L_{j,1} \vee L_{j,2} \vee L_{j,3}$ and each literal $L_{j,\ell}$ equals x or $\overline{x}$, with $x \in Vars(\varphi)$. For ease of notation, we assume $Vars(\varphi) = \{x_1, \ldots, x_m\}$, and we refer to the clauses in φ by $C_{m+1}, \ldots, C_n$, thus with the variables and clauses having indices over disjoint intervals.

Next, we construct a vertex-colored graph $G(\varphi)$ and show that $G(\varphi)$ contains a properly colored cycle iff φ is satisfiable.

For the construction, we distinguish the following gadgets, which are disjoint subgraphs of $G(\varphi)$:

- A *variable gadget* $G(x_i) = (V_i, E_i)$ for every variable x_i in φ with vertices and edges as depicted in Figure 5(a); the intuition is that $v_{i,1}$ encodes the choice to make x_i *true* and $v_{i,2}$ encodes the choice to make x_i *false.* A path from $v_{i,in}$ to $v_{i,out}$ then encodes the inverse truth assignment for x_i: x_i is assigned *true* iff the path visits vertex $v_{i,2}$.
- A *clause gadget* $G(C_j) = (V_j, E_j)$ for every clause C_j in φ with vertices and edges as depicted in Figure 5(b); the intuition is that vertices $v_{j,\ell}$ encode the literals $L_{j,\ell}$ in clause C_j. A path from $v_{j,in}$ to $v_{j,out}$ then encodes the choice of which literal in clause C_j is true.

Now, define $G(\varphi) = (V_\varphi, E_\varphi, K_\varphi, f_\varphi)$ as the following vertex-colored graph:

- $V_\varphi = \{v_0\} \cup V_1 \cup \cdots \cup V_n$ contains a special start vertex v_0 and the vertices necessary to describe gadgets $G(x_i)$ and $G(C_j)$ for every variable x_i and clause C_j in φ;
- E_φ consists of the following edges:
  - edges E_i and E_j from gadgets $G(x_i)$ and $G(C_j)$ for every variable x_i and clause C_j in φ;
  - edges from $v_{i,out}$ to $v_{i+1,in}$, for $i \in [1, n-1]$, to chain all variable gadgets and clause gadgets after one other;
  - edges $(v_0, v_{1,in})$ and $(v_{m,out}, v_0)$ to connect the chain with start node v_0 creating a cycle;
  - edges between variables in variable gadgets and their occurrence in clause gadgets:
    - ∗ an edge from each vertex $v_{i,1}$ in a variable gadget to each vertex $v_{j,\ell}$ in clause gadgets with $v_{j,\ell}$ representing a literal $L_{j,\ell} = x_i$ (recall that $v_{i,1}$ encodes *true* for x_i);
    - ∗ an edge from each vertex $v_{i,2}$ in variable gadgets to each vertex $v_{j,\ell}$ in clause gadgets where $v_{j,\ell}$ represents a literal $L_{j,\ell} = \overline{x}_i$ (recall that $v_{i,2}$ encodes *false* for x_i);

    We refer to these types of edges as *consistency edges* as appropriate coloring will ensure a consistent interpretation of the truth assignment.
- $K_\varphi = K_{\text{variable}} \cup K_{\text{other}}$ with
  - $K_{\text{variable}} = \{x_i, \overline{x}_i \mid x_i \in Vars(\varphi)\}$; and,
  - K_{other} a set of $|V_\varphi| - 3n + m$ colors distinct from K_{variable}.

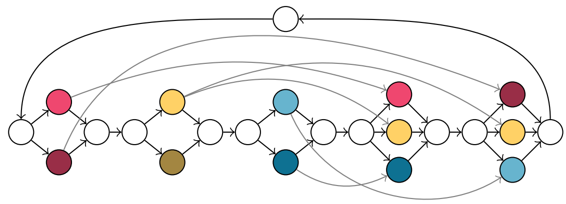


Fig. 6. $G(\varphi_1)$ for $\varphi_1 = (x_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3)$. For ease of exposition, vertices assigned with a unique color from K_{other} are left blank.

- f_φ is defined as follows:
  - $f_\varphi(v_{i,1}) = x_i$ and $f_\varphi(v_{i,2}) = \overline{x}_i$ for every $x_i \in Vars(\varphi)$;
  - $f_\varphi(v_{j,\ell}) = L_{j,\ell}$ for $j \in [m+1, n]$ and $\ell \in \{1,2,3\}$.
  - for all other vertices $v \in V_\varphi$, $f(v)$ is assigned a different color in K_{other}.

Example 27. Consider $\varphi_1 = (x_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3)$. Then $G(\varphi_1)$ is given in Figure 6.

The following lemma then implies NP-hardness.

LEMMA 28. *Let φ be a propositional logic formula in 3CNF. Then, φ is satisfiable iff $G(\varphi)$ has a properly colored cycle.*

PROOF. *(if)* Assume C is a properly colored cycle. By construction of $G(\varphi)$, a properly colored cycle always needs to go through each variable and clause gadget exactly once. Indeed, no cycle can use one of the shortcut consistency edges as the adjacent vertices carry the same color. Therefore, C picks for every variable x_i either the vertex $v_{i,1}$ enoding *true* or vertex $v_{i,2}$ encoding *false* in the variable gadget $G(x)$. Furthermore, in every clause gadget $G(C_j)$, C picks a single vertex $v_{j,\ell}$ encoding literal $L_{j,\ell}$ in C_j. Let ξ be the truth assignment that maps every variable x_i to *false* when $v_{i,1}$ is picked by C and to *true* when $v_{i,2}$ is picked. So, the choices of C represent the complement of the truth assignment. Notice, that under ξ every clause C_j evaluates to true. Indeed, let $L_{j,\ell}$ be the literal picked by C. When $L_{j,\ell} = x_i$ for some $x_i \in Vars(\varphi)$, then the vertices $v_{j,\ell}$ and $v_{x,1}$ in $G(\varphi)$ are connected with a consistency edge and are both labeled with the same color. As C is properly colored, this means that C must have picked the vertex $v_{i,2}$ and $\xi(x_i) = \xi(L_{j,\ell}) = true$. The same reasoning holds when $L_{j,\ell} = \overline{x}_i$. It thus follows that φ evaluates to true under ξ.

(only if). Let ξ be a satisfying truth assignment for φ. Then, let C be the path through $G(\varphi)$ that starts and ends in v_0 and that picks in every variable gadget $G(x_i)$, the vertex $v_{i,1}$ when $\xi(x_i)$ is *false* and $v_{i,2}$ otherwise. Furthermore, C picks in every clause gadget $G(C_j)$ a literal $L_{j,\ell}$ such that $\xi(L_{j,\ell})$ is *true*. The only possibility to violate properly coloring is through the consistency edges as these are the only edges where endpoints carry the same color. Assume two vertices $v_{i,1}$ (with $i \in [1,m]$) and $v_{j,\ell}$ (with $j \in [m+1,n]$) are picked by C that carry the same color. By construction, this color then is x_i meaning that $\xi(x_i) = false$ by assumption on the choice of C on variables. Furthermore, $\xi(C_{j,\ell}) = \xi(x_i) = true$ by assumption on the choice of C in clause gadgets. This leads to the desired contradiction. A similar argument can be made when $v_{i,2}$ and $v_{j,\ell}$ are picked by C. This concludes the proof. □

5.3 CONP-completeness

Next, we turn to the main result of this section showing that ROBUSTNESS(READ COMMITTED) is CONP-complete. The remainder of this section is devoted to the proof of the following theorem:

THEOREM 29. ROBUSTNESS*(*READ COMMITTED*) is* CONP*-complete.*

Obviously, ROBUSTNESS(READ COMMITTED) is in CONP. Indeed, for a set of transactions $\mathcal{T}$, just guess a counter-example schedule s over $\mathcal{T}$; then check that s is allowed under READ COMMITTED and that $CG(s)$ has a cycle. As the size of the guessed schedule is linear in the size of $\mathcal{T}$, and the checking step is in polynomial time, the latter procedure is in NP.

The remainder of this section is devoted to a reduction from the NP-complete 3SAT problem to the complement of ROBUSTNESS(READ COMMITTED), from which Theorem 29 then follows. For this, let φ be a Boolean formula in 3CNF given as input to 3SAT. Thus, φ is a conjunction of clauses C_j of the form $L_{j,1} \vee L_{j,2} \vee L_{j,3}$ with literals $L_{j,\ell}$ that either equal a variable x or a variable's complement $\overline{x}$, with $x \in Vars(\varphi)$. Analogously to Section 5.2, we assume $Vars(\varphi) = \{x_1, \ldots, x_m\}$ and refer to the clauses in φ by $C_{m+1}, \ldots, C_n$.

Next, we define a set $\mathcal{T}(\varphi)$ of transactions that (we will later show) is not robust under isolation level READ COMMITTED iff φ is satisfiable. The construction is similar to the construction of $G(\varphi)$ in the previous section. In fact, we construct $\mathcal{T}(\varphi)$ so to have exactly one transaction for every vertex in $G(\varphi)$. All transactions corresponding to vertices in (variable and clause) gadgets follow the following template ($\star$):

- write to a distinguished object that identifies the vertex under consideration;
- read the objects that identify the successor vertices; and,
- read all objects that identify the predecessor vertices.

When the transaction corresponds to an inner vertex of a gadget (a vertex of the form $v_{j,\ell}$ with $\ell \in [1, 3]$), the above template is preceded by writes to objects U_j^ℓ to deal with consistency edges.

A formal construction of $\mathcal{T}(\varphi)$ is given below. We omit defining **Obj** explicitly, as the necessary objects can be derived straightforwardly from the below transaction definitions. For ease of exposition we also omit $\mathsf{C_i}$ at the end of every transaction T_i.

For every variable x_i in φ, $\mathcal{T}(\varphi)$ contains a variable gadget $\mathcal{T}(\varphi, i)$ consisting of the following four transactions:

$$\begin{aligned}
T_{i,in} &: \mathsf{W_{i,in}[X_i], R_{i,in}[Y_i^1], R_{i,in}[Y_i^2], R_{i,in}[Z_{i-1}]},\\
T_{i,1} &: \textit{conflict-set}_{i,1}, \mathsf{W_{i,1}[Y_i^1], R_{i,1}[Z_i], R_{i,1}[X_i]},\\
T_{i,2} &: \textit{conflict-set}_{i,2}, \mathsf{W_{i,2}[Y_i^2], R_{i,2}[Z_i], R_{i,2}[X_i]},\\
T_{i,out} &: \mathsf{W_{i,out}[Z_i], R_{i,out}[X_{i+1}], R_{i,out}[Y_i^1], R_{i,out}[Y_i^2]}.
\end{aligned}$$

with *conflict-set*$_{i,1}$ and *conflict-set*$_{i,2}$ a sequence of write operations that will be specified later.

In this construction, $T_{i,in}$ and $T_{i,out}$, respectively, represent the *in*- and *out*-vertex of the variable gadget $G(x_i)$, that is, vertices $v_{i,in}$ and $v_{i,out}$, respectively. In addition, the transactions $T_{i,1}$ and $T_{i,2}$ represent the remaining two inner vertices $v_{i,1}$ and $v_{i,2}$, respectively. Notice, that these transactions correspond to the template ($\star$). Indeed, consider for instance the transaction $T_{i,in}$ corresponding to vertex $v_{i,in}$, which is identified by object X_i and who has successors $v_{i,1}$ and $v_{i,2}$ in $G(\varphi)$ represented by objects Y_i^1 and Y_i^2, respectively. Furthermore, $v_{i,in}$ has exactly one predecessor $v_{i-1,out}$ identified by Z_{i-1} when $i > 1$, and otherwise has v_0 as predecessor, which in turn is identified by object Z_0.

For every clause C_j in φ, we have a gadget $\mathcal{U}(\varphi, j)$ consisting of the following five transactions:

$$\begin{aligned}
T_{j,in} &: \mathsf{W_{j,in}[X_j], R_{j,in}[Y_j^1], R_{j,in}[Y_j^2], R_{j,in}[Y_j^3], R_{j,in}[Z_{j-1}]},\\
T_{j,1} &: \mathsf{W_{j,1}[U_j^1], W_{j,1}[Y_j^1], R_{j,1}[Z_j], R_{j,1}[X_j]},\\
T_{j,2} &: \mathsf{W_{j,2}[U_j^2], W_{j,2}[Y_j^2], R_{j,2}[Z_j], R_{j,2}[X_j]},\\
T_{j,3} &: \mathsf{W_{j,3}[U_j^3], W_{j,3}[Y_j^3], R_{j,3}[Z_j], R_{j,3}[X_j]},\\
T_{j,out} &: \mathsf{W_{j,out}[Z_j], R_{j,out}[X_{j+1}], R_{j,out}[Y_j^1], R_{j,out}[Y_j^2], R_{j,out}[Y_j^3]}.
\end{aligned}$$

In this construction, $T_{j,in}$ and $T_{j,out}$ represent the *in*- and *out*-vertex of the clause gadget $G(C_j)$. The transactions $T_{j,1}$, $T_{j,2}$, and $T_{j,3}$ represent the remaining three inner vertices of the clause gadget. Notice that the above transactions follow template (⋆) as well. Furthermore, every ℓth inner vertex has the additional identifier $\mathtt{U}_j^\ell$ that its corresponding transaction writes to.

Finally, $\mathcal{T}(\varphi)$ contains also the next transaction, corresponding to vertex v_0 in $G(\varphi)$:

$$T_0 : \mathtt{W}_0[\mathtt{Z}_0], \mathtt{R}_0[\mathtt{X}_1], \mathtt{W}_0[\mathtt{X}_{n+1}].$$

It remains to specify the conflict sets, whose purpose it is to represent the consistency edges in $G(\varphi)$. For $i \in [1, m]$, *conflict-set*$_{i,1}$ consists of all $\mathtt{W}_{i,1}[\mathtt{U}_j^\ell]$ such that $L_{j,\ell} = x_i$ in clause C_j for some $j \in [m+1, n]$ and $\ell \in \{1, 2, 3\}$. Similarly, *conflict-set*$_{i,2}$ consists of all $\mathtt{W}_{i,2}[\mathtt{U}_j^\ell]$ such that $L_{j,\ell} = \overline{x}_i$ in clause C_j for some $j \in [m+1, n]$ and $\ell \in \{1, 2, 3\}$. That is, every occurrence of variable x_i (respectively, $\overline{x}_i$) in the ℓth position of a clause C_j is witnessed by a write to $\mathtt{U}_j^\ell$ in $T_{i,1}$ (respectively, $T_{i,2}$).

Let $\beta : V_\varphi \leftrightarrow \mathcal{T}(\varphi)$ be the bijection that associates the vertices in $G(\varphi)$ with their corresponding transaction in $\mathcal{T}(\varphi)$. The following lemma details the correspondence between $T(\varphi)$ and $G(\varphi)$:

Lemma 30. *For every $v, v' \in V_\varphi$:*

(1) *$(v, v') \in E_\varphi$ implies there is an edge from $\beta(v)$ to $\beta(v')$ in the interference graph of $\mathcal{T}(\varphi)$; and,*
(2) *an edge from $\beta(v)$ to $\beta(v')$ in the interference graph of $\mathcal{T}(\varphi)$ implies either $(v, v') \in E_\varphi$ or $(v', v) \in E_\varphi$.*

Proof. (1) For every edge (v, v'), transactions $\beta(v)$ and $\beta(v')$ share an object that one writes to and the other reads from (or both write to, in the case that (v, v') represents a consistency edge, but this case is analogous). For example, for $v = v_{i,in}$ and $v' = v_{i,\ell}$ the corresponding transaction $T_{i,in}$ has operation $\mathtt{W}_{i,in}[\mathtt{X}_i]$ and $T_{i,\ell}$ has conflicting operation $\mathtt{R}_{i,\ell}[\mathtt{X}_i]$. The latter implies two edges in the interference graph, one from $T_{i,in}$ to $T_{i,\ell}$ and one from $T_{i,\ell}$ to $T_{i,in}$. It is easy to verify from the construction of $\mathcal{T}(\varphi)$ and bijection β that this observation is true for every edge (v, v') and thus that there is indeed an edge from $\beta(v)$ to $\beta(v')$ (and vice versa) in $IG(\mathcal{T}(\varphi))$.

(2) In the construction of $\mathcal{T}(\varphi)$, for every object, either no two transactions write to this object, or exactly two transactions write to this object, but no other transaction reads this object. Thus for transactions T and T' adjacent in $IG(\mathcal{T}(\varphi))$ one contains a write and the other a read or write to a common object. It is easy to verify from the construction that the only transactions T and T' with this property are transactions whose associated vertices $\beta^{-1}(T)$ and $\beta^{-1}(T')$ are indeed adjacent in $G(\varphi)$. Notice that if T and T' both contain a write to a common object, then the edge between $\beta^{-1}(T)$ and $\beta^{-1}(T')$ in $G(\varphi)$ is a consistency edge. □

As $T(\varphi)$ can be constructed in logspace, Theorem 29 then follows from Lemmas 31 and 32.

Lemma 31. *If there is a properly colored cycle in $G(\varphi)$, then $\mathcal{T}(\varphi)$ is not robust against* read committed.

Proof. Let C_φ be a properly colored cycle in $G(\varphi)$. As argued in the proof of Lemma 28, C_φ passes through the special vertex v_0 as well as through each variable and clause gadget in $G(\varphi)$. Let the following sequence be the result of applying β on the vertices in C_φ in the order as they appear in C_φ starting with v_0:

$$T_0, T_{1,in}, T_{1,k_1}, T_{1,out}, \ldots, T_{n,in}, T_{n,k_n}, T_{n,out}.$$

Denote the set consisting of all transactions in this sequence by $\mathcal{T}'$. By Lemma 30, there is a cycle $C_\mathcal{T}$ in $IG(\mathcal{T}(\varphi))$ that corresponds to C_φ. Then, $C_\mathcal{T}$ is transferable in T_0 on operations $(\mathtt{R}_0[\mathtt{X}_1], \mathtt{W}_0[\mathtt{X}_{n+1}])$.

Next, we construct a multi-split schedule for $\mathcal{T}'$. To this end, we introduce the following notation. Let $b_0 = \mathtt{R_0[X_1]}$ and let

$$b_{i,\alpha} = \begin{cases} \mathtt{R_{i,in}[Y_i^\ell]}, & \text{if } \alpha = in \text{ and } T_{i,\ell} \text{ follows } T_{i,in} \text{ in } C_\mathcal{T} \\ \mathtt{R_{i,\alpha}[Z_i]}, & \text{if } \alpha \in \{1,2,3\} \\ \mathtt{R_{i,out}[X_{i+1}]}, & \text{if } \alpha = out \end{cases}$$

for every $i \in [1, n]$. Clearly, $b_0 \in T_0$ and notice further that every $b_{i,\alpha}$ occurs in $T_{i,\alpha}$. For $i \in [1, n]$, denote by prefix_i the sequence

$$\text{prefix}_{b_{i,in}}(T_{i,in}), \text{prefix}_{b_{i,k_i}}(T_{i,k_i}), \text{prefix}_{b_{i,out}}(T_{i,out}),$$

and by postfix_i the sequence

$$\text{postfix}_{b_{i,in}}(T_{i,in}), \text{postfix}_{b_{i,k_i}}(T_{i,k_i}), \text{postfix}_{b_{i,out}}(T_{i,out}).$$

Now, let s' be the schedule over $\mathcal{T}'$ of the following form:

$$\text{prefix}_{b_0}(T_0) \cdot \text{prefix}_1 \cdots \text{prefix}_n \cdot \text{postfix}_{b_0}(T_0) \cdot \text{postfix}_1 \cdots \text{postfix}_n.$$

Notice that s' is indeed a multi-split schedule based on $C_\mathcal{T}$ on operations $(\mathtt{R_0[X_1]}, \mathtt{W_0[X_{n+1}]})$ (cf. Definition 18).

We argue that s' is allowed under READ COMMITTED. Recall from Definition 3 that s' is allowed under READ COMMITTED if it does not exhibit any dirty writes or dirty reads.

(s' exhibits no dirty writes). Note that by construction of $\mathcal{T}(\varphi)$, the only possible write-write conflicts in s' are between operations $\mathtt{W_{i,1}[U_j^\ell]}$ (or $\mathtt{W_{i,2}[U_j^\ell]}$) in $T_{i,1}$ (or $T_{i,2}$) and $\mathtt{W_{j,1}[U_j^\ell]}$ in $T_{j,\ell}$, with $i \in [1, m]$, $j \in [m+1, n]$, and $\ell \in \{1, 2, 3\}$, since all other objects occurring in $\mathcal{T}(\varphi)$ are written to by exactly one transaction.

Let $v_{i,k}$ and $v_{j,\ell}$ be the vertices in $G(\varphi)$ corresponding to these two transactions $T_{i,1}$ (or $T_{i,2}$) and $T_{j,\ell}$. By construction of $\mathcal{T}(\varphi)$, a write-write conflict between these two transactions implies that $v_{i,k}$ and $v_{j,\ell}$ are assigned the same color by f_φ, and $(v_{i,k}, v_{j,\ell}) \in E_\varphi$. Because of this, the properly colored cycle C_φ cannot contain both v_i and v_j, and hence $C_\mathcal{T}$ cannot contain both $T_{i,1}$ (or $T_{i,2}$) and $T_{j,\ell}$. We conclude that s' has no write-write conflicts and therefore exhibits no dirty writes.

(s' exhibits no dirty reads). We show that for every object $\mathtt{x} \in \bigcup\{\mathtt{X}_i, \mathtt{Y}_i^\ell, \mathtt{Z}_i \mid i \in [0, n+1], \ell \in \{1, 2, 3\}\}$ occurring in s' that if both $\mathtt{W_i[x]} \in T_i$ and $\mathtt{R_j[x]} \in T_j$ are in s' for some pair of transactions T_i and T_j, then $\mathtt{W_i[x]} <_{s'} \mathtt{C_i} <_{s'} \mathtt{R_j[x]}$ or $\mathtt{R_j[x]} <_{s'} \mathtt{W_i[x]}$. To this end, let $\mathtt{x} = \mathtt{X}_i$ for some $i \in [1, n]$ (the reasoning is analogous for $\mathtt{Y}_i^\ell$ and $\mathtt{Z}_i$). By construction of $\mathcal{T}(\varphi)$ and $C_\mathcal{T}$, the only transaction with a write operation on $\mathtt{X}_i$ is $T_{i,in}$, and the only transactions in s' reading this object are $T_{i-1,out}$ (or T_0 if $i = 1$, but this case is analogous) and $T_{i,\ell}$, for some $\ell \in \{1, 2, 3\}$. By construction of s' we have $\mathtt{R_{i-1,out}[X_i]} <_{s'} \mathtt{W_{i,in}[X_i]}$ in the former case and $\mathtt{W_{i,in}[X_i]} <_{s'} \mathtt{C_{i,in}} <_{s'} \mathtt{R_{i,\ell}[X_i]}$ in the latter case. Last, we consider the two write operations in T_0, namely $\mathtt{W_0[Z_0]}$ and $\mathtt{W_0[X_{n+1}]}$. The only transactions in s' with a conflicting read operation on these objects are $T_{1,in}$ and $T_{n,out}$, respectively. By construction, $\mathtt{W_0[Z_0]} <_{s'} \mathtt{C_0} <_{s'} \mathtt{R_{1,in}[Z_0]}$ and $\mathtt{R_{n,out}[X_{n+1}]} <_{s'} \mathtt{W_0[X_{n+1}]}$.

To conclude the proof, it suffices to remark that the transactions occurring in $\mathcal{T}(\varphi) \backslash \mathcal{T}'$ can be appended to s' in a serial fashion and in arbitrary order to obtain the required schedule s for $\mathcal{T}(\varphi)$ that is allowed under READ COMMITTED. Indeed, s is clearly still allowed under READ COMMITTED and has cycle $C_\mathcal{T}$ in its conflict graph. By Theorem 2, $\mathcal{T}(\varphi)$ is thus not robust against READ COMMITTED. □

Lemma 30(1) provides a direct way to obtain a set of transactions from a properly colored cycle thereby facilitating the proof of Lemma 31. The main difficulty in the proof of the next lemma stating the converse direction is that the interference graph for $T(\varphi)$ is bidirectional and can therefore contain cycles not corresponding to a cycle in $G(\varphi)$, which is problematic for the reduction.

Lemma 32. *If $\mathcal{T}(\varphi)$ is not robust against* read committed, *then there is a properly colored cycle in $G(\varphi)$.*

Proof. Assume $\mathcal{T}(\varphi)$ is not robust for read committed. According to Theorem 25, there exists a multi-split schedule s for $\mathcal{T}(\varphi)$ based on some transferable cycle $C_{\mathcal{T}}$ that is allowed under read committed. We argue that $C_{\mathcal{T}}$ corresponds to a properly colored cycle in $G(\varphi)$. To this end, we introduce some notation. For $i \in [1, n]\}$, let

$$\begin{array}{lcl}\omega_i^{in} & := & (T_{i,in}, b_{i,in}, a_{i,k_i}, T_{i,k_i});\\ \omega_i^{out} & := & (T_{i,k_i}, b_{i,k_i}, a_{i,out}, T_{i,out});\ \text{and,}\\ \omega_i^{\sim} & := & (T_{i,out}, b_{i,out}, a_{i+1,in}, T_{i+1,in}),\end{array}$$

where

$$b_{i,\alpha} = \begin{cases} \mathtt{R_{i,in}[Y_i^{\ell}]}, & \text{if } \alpha = in \text{ and } T_{i,\ell} \text{ follows } T_{i,in} \text{ in } C_{\mathcal{T}} \\ \mathtt{R_{i,\alpha}[Z_i]}, & \text{if } \alpha \in \{1,2,3\} \\ \mathtt{R_{i,out}[X_{i+1}]}, & \text{if } \alpha = out \end{cases}$$

and

$$a_{i,\alpha} = \begin{cases} \mathtt{W_{i,in}[X_i]}, & \text{if } \alpha = in \\ \mathtt{W_{i,\alpha}[Y_i^{\alpha}]}, & \text{if } \alpha \in \{1,2,3\} \\ \mathtt{W_{i,out}[Z_i]}, & \text{if } \alpha = out. \end{cases}$$

Furthermore, let $a_0 = \mathtt{W_{\emptyset}[X_{n+1}]}$, $b_0 = \mathtt{R_{\emptyset}[X_1]}$.

We prove the following two claims to be true:

(C1) The cycle $C_{\mathcal{T}}$ is transferable in T_0 on (b_0, a_0) and has the following form:

$$(T_0, b_0, a_{1,in}, T_{1,in}), \omega_1^{in}, \omega_1^{out}, \omega_1^{\sim}, \omega_2^{in}, \omega_2^{out}, \omega_2^{\sim},\\ \dots, \omega_{n-1}^{\sim}, \omega_n^{in}, \omega_n^{out}, (T_{n,out}, b_{n,out}, a_0, T_0).$$

(C2) The schedule s is fully split.

It follows immediately from Claim (C1) that $C_{\mathcal{T}}$ directly corresponds to a valid cycle C through each gadget in $G(\varphi)$; that is, edges are followed in the correct direction. Toward a contradiction, assume that C is not a properly colored cycle in $G(\varphi)$. Then, by construction, as similarly colored nodes are only connected through consistency edges, there are two transactions $T_{i,k}$ and $T_{j,\ell}$ with $i \in [1, m]$, $j \in [m+1, n]$, $k \in \{1, 2\}$ and $\ell \in \{1, 2, 3\}$, corresponding to the two vertices with the same color in respectively a variable gadget $G(x_i)$ and a clause gadget $G(C_j)$. In this case, both $T_{i,k}$ and $T_{j,\ell}$ contain a write operation on object U_j^{ℓ} in respectively $\text{prefix}_{b_{i,k}}(T_{i,k})$ and $\text{prefix}_{b_{j,\ell}}(T_{j,\ell})$. However, by Condition (C2) $\text{postfix}_{b_{i,k}}(T_{i,k})$ is not empty, implying that the conflicting write of $T_{j,\ell}$ happens after the write of $T_{i,k}$, but before the commit of $T_{i,k}$. As a result, s cannot be allowed under read committed, leading to the desired contradiction.

It remains to show that Claims (C1) and (C2) hold. To this end, we prove the correctness of some auxiliary conditions first. Henceforth, we often refer to gadgets by their index. We say that

$i \in [1, n]$ corresponds to a variable gadget if $i \leq m$ and that i corresponds to a clause gadget if $i > m$. Let ϵ be the function for s as defined in Definition 18.

(C32.3) For any variable gadget $\mathcal{T}(\varphi, i)$ with $i \in [1, m]$ and any $k \in \{1, 2\}$, if the cycle $C_\mathcal{T}$ contains an edge $(T_{i,k}, b_{i,k}, a_h, T_h)$ with $b_{i,k} \in \mathit{conflict\text{-}set}_{i,k}$ and a_h an arbitrary operation in some transaction T_h conflicting with $b_{i,k}$, then $\mathsf{postfix}_{\epsilon(T_{i,k})}(T_{i,k})$ is empty in s (that is, $T_{i,k}$ consists of one chunk in s).

Assume toward a contradiction that Condition (C32.3) does not hold, i.e., $b_{i,k} = \mathtt{W}_{\mathtt{i,k}}[\mathtt{U}_\mathtt{j}^\ell]$ and the $\mathsf{postfix}_{\epsilon(T_{i,k})}(T_{i,k})$ contains at least one operation in s (i.e., $\epsilon(T_{i,k}) = b_{i,k}$). It immediately follows that the next transaction in $C_\mathcal{T}$ (mentioned T_h in the condition) is $T_{j,\ell}$, since this is the only transaction containing an operation conflicting with $b_{i,k}$, namely $a_{j,\ell} = \mathtt{W}_{\mathtt{j},\ell}[\mathtt{U}_\mathtt{j}^\ell]$. If $T_{j,\ell}$ is not the first transaction in $C_\mathcal{T}$, then s looks as follows:

$$\ldots, \mathsf{prefix}_{b_{i,k}}(T_{i,k}), \mathsf{prefix}_{\epsilon(T_{j,\ell})}(T_{j,\ell}), \ldots, \mathsf{postfix}_{b_{i,k}}(T_{i,k}), \mathsf{postfix}_{\epsilon(T_{j,\ell})}(T_{j,\ell}), \ldots,$$

with $\epsilon(T_{j,\ell}) = \mathtt{C}_{\mathtt{j},\ell}$ or $\epsilon(T_{j,\ell}) = b_{j,\ell}$ for some operation $b_{j,\ell}$ in $T_{j,\ell}$. If instead $T_{j,\ell}$ is the first transaction in $C_\mathcal{T}$, then s looks as follows:

$$\mathsf{prefix}_{\epsilon(T_{j,\ell})}(T_{j,\ell}), \ldots, \mathsf{prefix}_{b_{i,k}}(T_{i,k}), \mathsf{postfix}_{\epsilon(T_{j,\ell})}(T_{j,\ell}), \ldots, \mathsf{postfix}_{b_{i,k}}(T_{i,k}), \ldots,$$

with $\epsilon(T_{j,\ell}) = b_{j,\ell}$ for some operation $b_{j,\ell}$ in $T_{j,\ell}$. Since $\mathtt{W}_{\mathtt{j},\ell}[\mathtt{U}_\mathtt{j}^\ell]$ is the first operation in $T_{j,\ell}$, it immediately follows that in both cases $\mathtt{W}_{\mathtt{j},\ell}[\mathtt{U}_\mathtt{j}^\ell] \in \mathsf{prefix}_{\epsilon(T_{j,\ell})}(T_{j,\ell})$, independent of our choice of $\epsilon(T_{j,\ell})$. In the former case, we therefore have $\mathtt{W}_{\mathtt{i,k}}[\mathtt{U}_\mathtt{j}^\ell] <_s \mathtt{W}_{\mathtt{j},\ell}[\mathtt{U}_\mathtt{j}^\ell] <_s \mathtt{C}_{\mathtt{i,k}}$. In the latter case, we have $\mathtt{W}_{\mathtt{j},\ell}[\mathtt{U}_\mathtt{j}^\ell] <_s \mathtt{W}_{\mathtt{i,k}}[\mathtt{U}_\mathtt{j}^\ell] <_s \mathtt{C}_{\mathtt{j},\ell}$ instead. As a result, in both cases s cannot be allowed under READ COMMITTED, leading to the desired contradiction.

(C32.4) . For any clause gadget $\mathcal{T}(\varphi, j)$ with $j \in [m+1, n]$ and any $k \in \{1, 2, 3\}$, if the cycle $C_\mathcal{T}$ contains an edge $(T_{j,k}, b_{j,k}, a_h, T_h)$ with $b_{j,k} = \mathtt{W}_{\mathtt{j,k}}[\mathtt{U}_\mathtt{j}^\mathtt{k}]$ and a_h an arbitrary operation in some transaction T_h conflicting with $b_{j,k}$, then $\mathsf{postfix}_{\epsilon(T_{j,k})}(T_{j,k})$ is empty in s, (that is, $T_{j,k}$ consists of one chunk in s).

Assume toward a contradiction that Condition (C32.4) does not hold, i.e., $b_{j,k} = \mathtt{W}_{\mathtt{j,k}}[\mathtt{U}_\mathtt{j}^\mathtt{k}]$ and $\mathsf{postfix}_{\epsilon(T_{j,k})}(T_{j,k})$ contains at least one operation in s, (i.e., $\epsilon(T_{j,k}) = b_{j,k}$). It immediately follows that the next transaction in $C_\mathcal{T}$ (mentioned T_h in the condition) is the transaction $T_{i,\ell}$ corresponding to a variable gadget $\mathcal{T}(\varphi, i)$ with $i \in [1, m]$ and with $a_{i,\ell} = \mathtt{W}_{\mathtt{i},\ell}[\mathtt{U}_\mathtt{j}^\mathtt{k}] \in \mathit{conflict\text{-}set}_{i,\ell}$, since this operation $a_{i,\ell}$ is the only other operation in $\mathcal{T}$ conflicting with $b_{j,k}$. If $T_{i,\ell}$ is the first transaction in $C_\mathcal{T}$, then s looks as follows:

$$\mathsf{prefix}_{b_{i,\ell}}(T_{i,\ell}), \ldots, \mathsf{prefix}_{b_{j,k}}(T_{j,k}), \mathsf{postfix}_{b_{i,\ell}}(T_{i,\ell}), \ldots, \mathsf{postfix}_{b_{j,k}}(T_{j,k}),$$

with $b_{i,\ell}$ an arbitrary operation before $a_{i,\ell}$ in $T_{i,\ell}$. As a result, we have $\mathtt{W}_{\mathtt{j,k}}[\mathtt{U}_\mathtt{j}^\mathtt{k}] <_s \mathtt{W}_{\mathtt{i},\ell}[\mathtt{U}_\mathtt{j}^\mathtt{k}] <_s \mathtt{C}_{\mathtt{j,k}}$, implying that s cannot be allowed under READ COMMITTED. If $T_{i,\ell}$ is not the first transaction in $C_\mathcal{T}$, then s looks as follows:

$$\ldots, \mathsf{prefix}_{b_{j,k}}(T_{j,k}), \mathsf{prefix}_{\epsilon(T_{i,\ell})}(T_{i,\ell}), \ldots, \mathsf{postfix}_{b_{j,k}}(T_{j,k}), \mathsf{postfix}_{\epsilon(T_{i,\ell})}(T_{i,\ell}), \ldots,$$

with $\epsilon(T_{i,\ell}) = \mathtt{C}_{\mathtt{i},\ell}$ or $\epsilon(T_{i,\ell}) = b_{i,\ell}$ for some arbitrary operation $b_{i,\ell}$ in $T_{i,\ell}$. However, if $a_{i,\ell} \in \mathsf{prefix}_{\epsilon(T_{i,\ell})}(T_{i,\ell})$, then we have $\mathtt{W}_{\mathtt{i},\ell}[\mathtt{U}_\mathtt{j}^\mathtt{k}] <_s \mathtt{W}_{\mathtt{j,k}}[\mathtt{U}_\mathtt{j}^\mathtt{k}] <_s \mathtt{C}_{\mathtt{i},\ell}$. We therefore conclude that s can only be allowed under READ COMMITTED if $\epsilon(T_{i,\ell}) = b_{i,\ell}$ with $b_{i,\ell} <_{T_{i,\ell}} a_{i,\ell}$ in $T_{i,\ell}$. By construction of $T_{i,\ell}$, we have $b_{i,\ell} \in \mathit{conflict\text{-}set}_{i,\ell}$. This observation contradicts with Condition (C32.3), since $\mathsf{postfix}_{b_{i,\ell}}(T_{i,\ell})$ is not empty, leading to the desired contradiction.

(C32.5). For any $i \in [1, n]$, transaction $T_{i,in}$ cannot be the first transaction in $C_\mathcal{T}$.

Assume toward a contradiction that Condition (C32.5) does not hold, i.e., $T_{i,in}$ is the first transaction in $C_{\mathcal{T}}$, implying that $C_{\mathcal{T}}$ is transferable in $T_{i,in}$ on $(b_{i,in}, a_{i,in})$ for some choice of operations $b_{i,in}$ and $a_{i,in}$ in $T_{i,in}$. Recall that the latter particulary means that $\epsilon(T_{i,in}) = b_{i,in}$. If $b_{i,in} = \mathtt{W_{i,in}[X_i]}$, then the transaction following $T_{i,in}$ in $C_{\mathcal{T}}$ is either $T_{i,k}$ for some $k \in \{1,2,3\}$, or $T_{i-1,out}$ (or T_0, in the special case that $i = 0$, but this case is analogous to $T_{i-1,out}$), since these transactions are the only ones containing a conflicting operation on object X_i. In the former case, the schedule s would look as follows:

$$\text{prefix}_{b_{i,in}}(T_{i,in}), \text{prefix}_{\epsilon(T_{i,k})}(T_{i,k}), \ldots, \text{postfix}_{b_{i,in}}(T_{i,in}), \text{postfix}_{\epsilon(T_{i,k})}(T_{i,k}), \ldots,$$

with $\epsilon(T_{i,k}) = b_{i,k}$ for some operation $b_{i,k}$ before $\mathtt{R_{i,k}[X_i]}$ in $T_{i,k}$, as otherwise we would have $\mathtt{W_{i,in}[X_i]} <_s \mathtt{R_{i,k}[X_i]} <_s \mathtt{C_{i,in}}$. Consequently, $\mathtt{W_{i,k}[Y_i^k]} \in \text{postfix}_{b_{i,k}}(T_{i,k})$, as otherwise $\mathtt{W_{i,k}[Y_i^k]} <_s \mathtt{R_{i,in}[Y_i^k]} <_s \mathtt{C_{i,k}}$. This implies, however, that $b_{i,k} \in \textit{conflict-set}_{i,k}$ or $b_{i,k} = \mathtt{W_{i,k}[U_i^k]}$, depending on whether $T_{i,in}$ is in a variable or clause gadget, thereby contradicting respectively Conditions (C32.3) or (C32.4). If, however, the next transaction in $C_{\mathcal{T}}$ is $T_{i-1,out}$, then we obtain by analogous reasoning that $\mathtt{W_{i-1,out}[Z_{i-1}]} \in \text{postfix}_{b_{i-1,out}}(T_{i-1,out})$. This is, however, impossible, since $\mathtt{W_{i-1,out}[Z_{i-1}]}$ is the first operation in $T_{i-1,out}$.

If $b_{i,in} = \mathtt{R_{i,in}[Y_i^k]}$, for some $k \in \{1,2,3\}$, then $a_{i,in} = \mathtt{R_{i,in}[Y_i^\ell]}$, for some $\ell \in 1,2,3$ with $k < \ell$, or $a_{i,in} = \mathtt{R_{i,in}[Z_{i-1}]}$. If $a_{i,in} = \mathtt{R_{i,in}[Y_i^\ell]}$, then the transaction preceding $T_{i,in}$ in $C_{\mathcal{T}}$ is $T_{i,\ell}$, since this is the only transaction containing an operation conflicting with $a_{i,in}$, namely $b_{i,\ell} = \mathtt{W_{i,\ell}[Y_i^\ell]}$. Hence, s looks as follows:

$$\text{prefix}_{b_{i,in}}(T_{i,in}), \ldots, \text{prefix}_{\epsilon(T_{i,\ell})}(T_{i,\ell}), \text{postfix}_{b_{i,in}}(T_{i,in}), \ldots, \text{postfix}_{\epsilon(T_{i,\ell})}(T_{i,\ell}),$$

with $\epsilon(T_{i,\ell}) = b_{i,\ell}$ or $\epsilon(T_{i,\ell}) = \mathtt{C_{i,\ell}}$. In the former case, we have $\mathtt{W_{i,\ell}[Y_i^\ell]} <_s \mathtt{R_{i,in}[Y_i^\ell]} <_s \mathtt{C_{i,\ell}}$. In the later case, we have $\mathtt{W_{i,in}[X_i]} <_s \mathtt{R_{i,\ell}[X_i]} <_s \mathtt{C_{i,in}}$. We conclude that in both cases s cannot be allowed under READ COMMITTED. If $a_{i,in} = \mathtt{R_{i,in}[Z_{i-1}]}$, then we analogously obtain that the transaction preceding $T_{i,in}$ in $C_{\mathcal{T}}$ is $T_{i-1,out}$, with conflicting operation $b_{i-1,out} = \mathtt{W_{i-1,out}[Z_{i-1}]}$, and hence s is as follows:

$$\text{prefix}_{b_{i,in}}(T_{i,in}), \ldots, \text{prefix}_{\epsilon(T_{i-1,out})}(T_{i-1,out}), \text{postfix}_{b_{i,in}}(T_{i,in}), \ldots, \text{postfix}_{\epsilon(T_{i-1,out})}(T_{i-1,out}),$$

with $\epsilon(T_{i-1,out}) = b_{i-1,out}$ or $\epsilon(T_{i-1,out}) = \mathtt{C_{i-1,out}}$. Analogously, we have $\mathtt{W_{i-1,out}[Z_{i-1}]} <_s \mathtt{R_{i,in}[Z_{i-1}]} <_s \mathtt{C_{i-1,out}}$ in the former case and $\mathtt{W_{i,in}[X_i]} <_s \mathtt{R_{i-1,out}[X_i]} <_s \mathtt{C_{i,in}}$ in the latter case. As a result, we cannot take $b_{i,in} = \mathtt{R_{i,in}[Y_i^k]}$ for some $k \in \{1,2,3\}$ if $T_{i,in}$ is the first transaction in $C_{\mathcal{T}}$.

Since $b_{i,in} <_{T_{i,in}} a_{i,in}$, this operation $b_{i,in}$ cannot be the last operation in $T_{i,in}$. We conclude that no suitable $b_{i,in}$ exists in $T_{i,in}$, leading to the desired contradiction.

(C32.6) For any $i \in [1,n]$ and any $k \in \{1,2\}$ (if i corresponds to a variable gadget in G) or $k \in \{1,2,3\}$ (if i corresponds to a clause gadget in G), transaction $T_{i,k}$ cannot be the first transaction in $C_{\mathcal{T}}$.

Assume toward a contradiction that Condition (C32.6) does not hold, i.e., $T_{i,k}$ is the first transaction in $C_{\mathcal{T}}$, implying that $C_{\mathcal{T}}$ is transferable in $T_{i,k}$ on $(b_{i,k}, a_{i,k})$ for some choice of operations $b_{i,k}$ and $a_{i,k}$ in $T_{i,k}$. Note that $b_{i,k}$ cannot be in $\textit{conflict-set}_{i,k}$ (if i corresponds to a variable gadget in G) or equal to $\mathtt{W_{i,k}[U_i^k]}$ (if i corresponds to a clause gadget in G), as this would contradict Conditions (C32.3) or (C32.4), respectively.

If $b_{i,k} = \mathtt{W_{i,k}[Y_i^k]}$, then the transaction following $T_{i,k}$ in $C_{\mathcal{T}}$ is either $T_{i,in}$ or $T_{i,out}$, since these transactions are the only ones containing a conflicting operation on object Y_i^k. In the former case, the schedule s would look as follows:

$$\text{prefix}_{b_{i,k}}(T_{i,k}), \text{prefix}_{\epsilon(T_{i,in})}(T_{i,in}), \dots, \text{postfix}_{b_{i,k}}(T_{i,k}), \text{postfix}_{\epsilon(T_{i,in})}(T_{i,in}), \dots,$$

with $\epsilon(T_{i,in}) = b_{i,in}$ for some operation $b_{i,in}$ before $\mathtt{R_{i,in}[Y_i^k]}$ in $T_{i,in}$, as otherwise we would have $\mathtt{W_{i,k}[Y_i^k]} <_s \mathtt{R_{i,in}[Y_i^k]} <_s \mathtt{C_{i,k}}$. Consequently, $\mathtt{W_{i,in}[X_i]} <_s \mathtt{R_{i,k}[X_i]} <_s \mathtt{C_{i,in}}$, unless $\mathtt{W_{i,in}[X_i]} \in \text{postfix}_{b_{i,in}}(T_{i,in})$. Since $\mathtt{W_{i,in}[X_i]}$ is the first operation of $T_{i,in}$, this is impossible. If, however, the next transaction in $C_{\mathcal{T}}$ is $T_{i,out}$, then we obtain by analogous reasoning that $\mathtt{W_{i,out}[Z_i]} \in \text{postfix}_{b_{i,out}}(T_{i,out})$. This is, however, impossible as well, since $\mathtt{W_{i,out}[Z_i]}$ is the first operation in $T_{i,out}$.

If $b_{i,k} = \mathtt{R_{i,k}[Z_i]}$, then $a_{i,k} = \mathtt{R_{i,k}[X_i]}$. Then, the transaction preceding $T_{i,k}$ in $C_{\mathcal{T}}$ is $T_{i,in}$, since this is the only transaction containing an operation conflicting with $a_{i,k}$, namely $b_{i,in} = \mathtt{W_{i,in}[X_i]}$. Hence, s looks as follows:

$$\text{prefix}_{b_{i,k}}(T_{i,k}), \dots, \text{prefix}_{\epsilon(T_{i,in})}(T_{i,in}), \text{postfix}_{b_{i,k}}(T_{i,k}), \dots, \text{postfix}_{\epsilon(T_{i,in})}(T_{i,in}),$$

with $\epsilon(T_{i,in}) = b_{i,in}$ or $\epsilon(T_{i,in}) = \mathtt{C_{i,in}}$. In the former case, we have $\mathtt{W_{i,in}[X_i]} <_s \mathtt{R_{i,k}[X_i]} <_s \mathtt{C_{i,in}}$. In the latter case, we have $\mathtt{W_{i,k}[Y_i^k]} <_s \mathtt{R_{i,in}[Y_i^k]} <_s \mathtt{C_{i,k}}$. As a result, we cannot take $b_{i,k} = \mathtt{R_{i,k}[Z_i]}$ if $T_{i,k}$ is the first transaction in $C_{\mathcal{T}}$.

Since $b_{i,k} <_{T_{i,k}} a_{i,k}$, this operation $b_{i,k}$ cannot be the last operation in $T_{i,k}$. We conclude that no suitable $b_{i,k}$ exists in $T_{i,k}$, leading to the desired contradiction.

(C32.7) For any $i \in [1, n]$, transaction $T_{i,out}$ cannot be the first transaction in $C_{\mathcal{T}}$.

Assume toward a contradiction that Condition (C32.7) does not hold, i.e., $T_{i,out}$ is the first transaction in $C_{\mathcal{T}}$, implying that $C_{\mathcal{T}}$ is transferable in $T_{i,out}$ on $(b_{i,out}, a_{i,out})$ for some choice of operations $b_{i,out}$ and $a_{i,out}$ in $T_{i,out}$.

If $b_{i,out} = \mathtt{W_{i,out}[Z_i]}$, then the transaction following $T_{i,out}$ in $C_{\mathcal{T}}$ is either $T_{i,k}$ for some $k \in \{1, 2, 3\}$ or $T_{i+1,in}$, since these transactions are the only ones containing a conflicting operation on object Z_i. In the former case, the schedule s would look as follows:

$$\text{prefix}_{b_{i,out}}(T_{i,out}), \text{prefix}_{\epsilon(T_{i,k})}(T_{i,k}), \dots, \text{postfix}_{b_{i,out}}(T_{i,out}), \text{postfix}_{\epsilon(T_{i,k})}(T_{i,k}), \dots,$$

with $\epsilon(T_{i,k}) = b_{i,k}$ for some operation $b_{i,k}$ before $\mathtt{R_{i,k}[Z_i]}$ in $T_{i,k}$, as otherwise we would have $\mathtt{W_{i,out}[Z_i]} <_s \mathtt{R_{i,k}[Z_i]} <_s \mathtt{C_{i,out}}$. Consequently, $\mathtt{W_{i,k}[Y_i^k]} <_s \mathtt{R_{i,out}[Y_i^k]} <_s \mathtt{C_{i,k}}$, unless $\mathtt{W_{i,in}[X_i]} \in \text{postfix}_{b_{i,in}}(T_{i,in})$. This implies that $b_{i,k} \in \textit{conflict-set}_{i,k}$ (if i corresponds to a variable gadget in G), or $b_{i,k} = \mathtt{W_{i,k}[U_i^k]}$ (if i corresponds to a clause gadget), thereby contradicting respectively Condition (C32.3) or (C32.4). If, however, the next transaction in $C_{\mathcal{T}}$ is $T_{i+1,in}$, then we obtain by analogous reasoning that $\mathtt{W_{i+1,in}[X_{i+1}]} \in \text{postfix}_{b_{i+1,in}}(T_{i+1,in})$. This is, however, impossible, since $\mathtt{W_{i+1,in}[X_{i+1}]}$ is the first operation in $T_{i+1,in}$.

If $b_{i,out} = \mathtt{R_{i,out}[X_{i+1}]}$ or $b_{i,out} = \mathtt{R_{i,out}[Y_i^k]}$ for some $k \in \{1, 2, 3\}$, then $a_{i,out} = \mathtt{R_{i,out}[Y_i^\ell]}$ for some $\ell \in \{1, 2, 3\}$, with $b_{i,out} <_{T_{i,out}} a_{i,out}$. Then, the transaction preceding $T_{i,out}$ in $C_{\mathcal{T}}$ is $T_{i,\ell}$, since this is the only transaction containing an operation conflicting with $a_{i,out}$, namely $b_{i,\ell} = \mathtt{W_{i,\ell}[Y_i^\ell]}$. Hence, s looks as follows:

$$\text{prefix}_{b_{i,out}}(T_{i,out}), \dots, \text{prefix}_{\epsilon(T_{i,\ell})}(T_{i,\ell}), \text{postfix}_{b_{i,out}}(T_{i,out}), \dots, \text{postfix}_{\epsilon(T_{i,\ell})}(T_{i,\ell}),$$

with $\epsilon(T_{i,\ell}) = b_{i,\ell}$ or $\epsilon(T_{i,\ell}) = \mathtt{C_{i,\ell}}$. In the former case, we have $\mathtt{W_{i,\ell}[Y_i^\ell]} <_s \mathtt{R_{i,out}[Y_i^\ell]} <_s \mathtt{C_{i,\ell}}$. In the latter case, we have $\mathtt{W_{i,out}[Z_i]} <_s \mathtt{R_{i,\ell}[Z_i]} <_s \mathtt{C_{i,out}}$. As a result, we cannot find a suitable $b_{i,out}$ if $T_{i,out}$ is the first transaction in $C_{\mathcal{T}}$, leading to the desired contradiction.

(C32.8) Transaction T_0 is the first transaction in $C_{\mathcal{T}}$ and $C_{\mathcal{T}}$ is transferable in T_0 on operations (b_0, a_0) with $b_0 = \mathtt{R_\emptyset[X_1]}$ and $a_0 = \mathtt{W_\emptyset[X_{n+1}]}$.

Assume toward a contradiction that Condition (C32.8) does not hold. It then follows from Conditions (C32.5)–(C32.7) that T_0 is the first transaction, but a_0 and/or b_0 are chosen differently. Since $b_0 <_{T_0} a_0$, there are two remaining options for a_0 and b_0:

- $b_0 = \mathtt{W_0[Z_0]}$ and $a_0 = \mathtt{R_0[X_1]}$, or
- $b_0 = \mathtt{W_0[Z_0]}$ and $a_0 = \mathtt{W_0[X_{n+1}]}$.

In both cases, there is only one operation conflicting with $b_0 = \mathtt{W_0[Z_0]}$ in T_0, namely $a_{1,in} = \mathtt{R_{1,in}[Z_0]}$ in $T_{1,in}$. The multi-split schedule s based on this cycle hence looks as follows:

$$\mathsf{prefix}_{b_0}(T_0), \mathsf{prefix}_{\epsilon(T_{1,in})}(T_{1,in}), \ldots, \mathsf{postfix}_{b_0}(T_0), \mathsf{postfix}_{\epsilon(T_{1,in})}(T_{1,in}), \ldots,$$

with $\epsilon(T_{1,in}) = \mathtt{C_{1,in}}$ or $\epsilon(T_{1,in}) = b_{1,in}$ for some operation $b_{1,in}$ in $T_{1,in}$. Note in particular that $\mathtt{W_{1,in}[X_1]}$ is always in $\mathsf{prefix}_{\epsilon(T_{1,in})}(T_{1,in})$, as otherwise $\mathsf{prefix}_{\epsilon(T_{1,in})}(T_{1,in})$ would be empty. However, if $\epsilon(T_{1,in}) = \mathtt{C_{1,in}}$, then $\mathtt{W_0[Z_0]} <_s \mathtt{R_{1,in}[Z_0]} <_s \mathtt{C_0}$, and if $\epsilon(T_{1,in}) = b_{1,in}$, then $\mathtt{W_{1,in}[X_1]} <_s \mathtt{R_0[X_0]} <_s \mathtt{C_{1,in}}$, contradicting in both cases our assumption that s is allowed under READ COMMITTED.

Henceforth, by Condition (C32.8), we will implicitly assume that T_0 is the first transaction in $C_{\mathcal{T}}$.

(C32.9) For every $i \in [2, n]$, if $C_{\mathcal{T}}$ contains the edge $(T_{i-1,out}, \mathtt{R_{i-1,out}[X_i]}, \mathtt{W_{i,in}[X_i]}, T_{i,in})$ and $\mathsf{postfix}_{\epsilon(T_{i,in})}(T_{i,in})$ is nonempty, then the next edge in $C_{\mathcal{T}}$ is $(T_{i,in}, \mathtt{R_{i,in}[Y_i^k]}, \mathtt{W_{i,k}[Y_i^k]}, T_{i,k})$ for some $k \in \{1, 2, 3\}$ and $\mathsf{postfix}_{\epsilon(T_{i,k})}(T_{i,k})$ is nonempty as well.

By definition of $C_{\mathcal{T}}$, the edge following $(T_{i-1,out}, \mathtt{R_{i-1,out}[X_i]}, \mathtt{W_{i,in}[X_i]}, T_{i,in})$ is of the form $(T_{i,in}, b_{i,in}, a_j, T_j)$, for some operation $b_{i,in}$ in $T_{i,in}$, and some operation a_j in T_j conflicting with $b_{i,in}$. Since $\mathsf{postfix}_{\epsilon(T_{i,in})}(T_{i,in})$ is nonempty, the schedule s looks as follows:

$$\ldots, \mathsf{prefix}_{b_{i,in}}(T_{i,in}), \mathsf{prefix}_{\epsilon(T_j)}(T_j), \ldots, \mathsf{postfix}_{b_{i,in}}(T_{i,in}), \mathsf{postfix}_{\epsilon(T_j)}(T_j), \ldots,$$

with $\epsilon(T_j) = \mathtt{C_j}$ or $\epsilon(T_j) = b_j$ for some operation b_j in T_j.

If $b_{i,in} = \mathtt{W_{i,in}[X_i]}$, then $T_j = T_{i,k}$ for some $k \in \{1, 2, 3\}$, since these transactions are the only transactions conflicting with $b_{i,in}$ (except of course $T_{i-1,out}$, but we cannot use the same transaction multiple times in $C_{\mathcal{T}}$). Note that $\mathtt{R_{i,k}[X_i]}$ needs to be in $\mathsf{postfix}_{\epsilon(T_{i,k})}(T_{i,k})$, as otherwise s would not be allowed under READ COMMITTED. However, in this case, $\mathtt{W_{i,k}[Y_i^k]}$ needs to be in $\mathsf{postfix}_{\epsilon(T_{i,k})}(T_{i,k})$ as well, as otherwise we have $\mathtt{W_{i,k}[Y_i^k]} <_s \mathtt{R_{i,in}[Y_i^k]} <_s \mathtt{C_{i,k}}$. But then $\epsilon(T_{i,k}) = b_{i,k}$ with $b_{i,k}$ either in *conflict-set*$_{i,k}$ (if i corresponds to a variable gadget in G) or equal to $\mathtt{W_{i,k}[U_i^k]}$ (if i corresponds to a clause gadget in G), thereby contradicting Condition (C32.3) or (C32.4), respectively.

Since $\mathsf{postfix}_{b_{i,in}}(T_{i,in})$ is not empty, $b_{i,in}$ cannot be the last operation in $T_{i,in}$. We conclude that $b_{i,in} = \mathtt{R_{i,in}[Y_i^k]}$ for some $k \in \{1, 2, 3\}$. It now follows immediately that the next edge is indeed $(T_{i,in}, \mathtt{R_{i,in}[Y_i^k]}, \mathtt{W_{i,k}[Y_i^k]}, T_{i,k})$, since $\mathtt{W_{i,k}[Y_i^k]}$ is the only operation conflicting with $b_{i,in}$. Furthermore, if s is allowed under READ COMMITTED, then $\mathtt{R_{i,k}[X_i]}$ is in $\mathsf{postfix}_{b_{i,k}}(T_{i,k})$, independent of our choice of $b_{i,k}$, as otherwise $\mathtt{W_{i,in}[X_i]} <_s \mathtt{R_{i,k}[X_i]} <_s \mathtt{C_{i,in}}$.

(C32.10) For every $i \in [1, n]$ and every $k \in \{1, 2, 3\}$, if $C_{\mathcal{T}}$ contains the edge $(T_{i,in}, \mathtt{R_{i,in}[Y_i^k]}, \mathtt{W_{i,k}[Y_i^k]}, T_{i,k})$ and $\mathsf{postfix}_{\epsilon(T_{i,k})}(T_{i,k})$ is nonempty, then the next edge in $C_{\mathcal{T}}$ is $(T_{i,k}, \mathtt{R_{i,k}[Z_i]}, \mathtt{W_{i,out}[Z_i]}, T_{i,out})$ and $\mathsf{postfix}_{\epsilon(T_{i,out})}(T_{i,out})$ is nonempty as well.

By definition of $C_{\mathcal{T}}$, the edge following $(T_{i,in}, \mathtt{R_{i,in}[Y_i^k]}, \mathtt{W_{i,k}[Y_i^k]}, T_{i,k})$ is of the form $(T_{i,k}, b_{i,k}, a_j, T_j)$, for some operation $b_{i,k}$ in $T_{i,k}$, and some operation a_j in T_j conflicting with $b_{i,k}$. Since the chunk

$\text{postfix}_{\epsilon(T_{i,k})}(T_{i,k})$ is nonempty, the schedule s looks as follows:

$$\ldots, \text{prefix}_{b_{i,k}}(T_{i,k}), \text{prefix}_{\epsilon(T_j)}(T_j), \ldots, \text{postfix}_{b_{i,k}}(T_{i,k}), \text{postfix}_{\epsilon(T_j)}(T_j), \ldots,$$

with $\epsilon(T_j) = \mathtt{C_j}$ or $\epsilon(T_j) = b_j$ for some operation b_j in T_j.

If i represents a variable gadget, then we cannot pick $b_{i,k} \in$ *conflict-set*$_{i,k}$, as this would contradict Condition (C32.3). Analogously, by Condition (C32.4), we cannot have $b_{i,k} = \mathtt{W_{i,k}[U_i^k]}$ if i represents a clause gadget.

If $b_{i,k} = \mathtt{W_{i,k}[Y_i^k]}$, then $T_j = T_{i,out}$, since this transaction is the only transaction conflicting with $b_{i,k}$ (except of course $T_{i,in}$, but we cannot use the same transaction multiple times in $C_{\mathcal{T}}$). Note that $\mathtt{R_{i,out}[Y_i^k]}$ needs to be in $\text{postfix}_{\epsilon(T_{i,out})}(T_{i,out})$, as otherwise s would not be allowed under READ COMMITTED. However, in this case, $\mathtt{W_{i,out}[Z_i]}$ needs to be in $\text{postfix}_{\epsilon(T_{i,out})}(T_{i,out})$ as well, as otherwise we have $\mathtt{W_{i,out}[Z_i]} <_s \mathtt{R_{i,k}[Z_i]} <_s \mathtt{C_{i,out}}$. Since $\mathtt{W_{i,out}[Z_i]}$ is the first operation in $T_{i,out}$, this cannot hold.

Since $\text{postfix}_{b_{i,k}}(T_{i,k})$ is not empty, $b_{i,k}$ cannot be the last operation in $T_{i,k}$. So our only remaining option is $b_{i,k} = \mathtt{R_{i,k}[Z_i]}$. It now follows immediately that the next edge is indeed $(T_{i,k}, \mathtt{R_{i,k}[Z_i]}, \mathtt{W_{i,out}[Z_i]}, T_{i,out})$, since $\mathtt{W_{i,out}[Z_i]}$ is the only operation conflicting with $b_{i,k}$. Furthermore, if s is allowed under READ COMMITTED, then $\mathtt{R_{i,out}[Y_i^k]}$ is in $\text{postfix}_{b_{i,out}}(T_{i,out})$, independent of our choice of $b_{i,out}$, as otherwise $\mathtt{W_{i,k}[Y_i^k]} <_s \mathtt{R_{i,out}[Y_i^k]} <_s \mathtt{C_{i,k}}$.

(C32.11) For every $i \in [1, n-1]$ and every $k \in \{1,2,3\}$, if $C_{\mathcal{T}}$ contains the edge $(T_{i,k}, \mathtt{R_{i,k}[Z_i]}, \mathtt{W_{i,out}[Z_i]}, T_{i,out})$ and $\text{postfix}_{\epsilon(T_{i,out})}(T_{i,out})$ is nonempty, then the next edge in $C_{\mathcal{T}}$ is $(T_{i,out}, \mathtt{R_{i,out}[X_{i+1}]}, \mathtt{W_{i+1,in}[X_{i+1}]}, T_{i+1,in})$ and $\text{postfix}_{\epsilon(T_{i+1,in})}(T_{i+1,in})$ is nonempty as well.

From the definition of $C_{\mathcal{T}}$, the edge following $(T_{i,k}, \mathtt{R_{i,k}[Z_i]}, \mathtt{W_{i,out}[Z_i]}, T_{i,out})$ is of the form $(T_{i,out}, b_{i,out}, a_j, T_j)$, for some operation $b_{i,out}$ in $T_{i,out}$, and some operation a_j in T_j conflicting with $b_{i,out}$. Since $\text{postfix}_{\epsilon(T_{i,out})}(T_{i,out})$ is nonempty, the schedule s looks as follows:

$$\ldots, \text{prefix}_{b_{i,out}}(T_{i,out}), \text{prefix}_{\epsilon(T_j)}(T_j), \ldots, \text{postfix}_{b_{i,out}}(T_{i,out}), \text{postfix}_{\epsilon(T_j)}(T_j), \ldots,$$

with $\epsilon(T_j) = \mathtt{C_j}$ or $\epsilon(T_j) = b_j$ for some operation b_j in T_j.

If $b_{i,out} = \mathtt{W_{i,out}[Z_i]}$, then T_j is either $T_{i+1,in}$ or $T_{i,\ell}$ for some $\ell in\{1,2,3\}$, since these transactions are the only transactions conflicting with $b_{i,out}$. If $T_j = T_{i+1,in}$,then $\mathtt{R_{i+1,in}[Z_i]}$ needs to be in $\text{postfix}_{\epsilon(T_{i+1,in})}(T_{i+1,in})$, as otherwise s would not be allowed under READ COMMITTED. However, in this case, $\mathtt{W_{i+1,in}[X_{i+1}]}$ needs to be in $\text{postfix}_{\epsilon(T_{i+1,in})}(T_{i+1,in})$ as well, as otherwise we have $\mathtt{W_{i+1,in}[X_{i+1}]} <_s \mathtt{R_{i,out}[X_{i+1}]} <_s \mathtt{C_{i+1,in}}$. Since $\mathtt{W_{i+1,in}[X_i]}$ is the first operation in $T_{i+1,in}$, this cannot hold. Analogously, if instead $T_j = T_{i,\ell}$, then $\mathtt{R_{i,\ell}[Z_i]}$ and consequently $\mathtt{W_{i,\ell}[Y_i^\ell]}$ need to be in $\text{postfix}_{\epsilon(T_{i,\ell})}(T_{i,\ell})$. Our only remaining option in this case is to pick $b_{i,\ell} \in$ *conflict-set*$_{i,\ell}$ (if i corresponds to a variable gadget in G), or $b_{i,\ell} = \mathtt{W_{i,\ell}[U_i^\ell]}$ (if i corresponds to a clause gadget in G), Thereby contradicting Conditions (C32.3) or (C32.4), respectively. Hence, we conclude that we cannot have $b_{i,out} = \mathtt{W_{i,out}[Z_i]}$.

If $b_{i,out} = \mathtt{R_{i,out}[Y_i^\ell]}$ for some $\ell \in \{1,2,3\}$ different from k, then $T_j = T_{i,\ell}$, and s looks as follows:

$$\ldots, \text{prefix}_{b_{i,k}}(T_{i,k}), \text{prefix}_{b_{i,out}}(T_{i,out}), \text{prefix}_{b_{i,\ell}}(T_{i,\ell}), \ldots,$$
$$\text{postfix}_{b_{i,k}}(T_{i,k}), \text{postfix}_{b_{i,out}}(T_{i,out}), \text{postfix}_{b_{i,\ell}}(T_{i,\ell}), \ldots,$$

with $b_{i,k} = \mathtt{R_{i,k}[Z_i]}$ and $b_{i,\ell}$ an operation in $T_{i,\ell}$ before $\mathtt{R_{i,\ell}[Z_i]}$. Indeed, otherwise we would have $\mathtt{W_{i,out}[Z_i]} <_s \mathtt{R_{i,\ell}[Z_i]} <_s \mathtt{C_{i,out}}$. Once again, by Conditions (C32.3) and (C32.4), our only option for $b_{i,\ell}$ is $\mathtt{W_{i,\ell}[Y_i^\ell]}$. This operation only conflicts with the operations $\mathtt{R_{i,out}[Y_i^\ell]}$ and $\mathtt{R_{i,in}[Y_i^\ell]}$ in respectively $T_{i,out}$ and $T_{i,in}$. Since $T_{i,out}$ cannot appear multiple times in $C_{\mathcal{T}}$, our only choice now is $T_{i,in}$ to continue the cycle. However, since $\mathtt{R_{i,k}[X_i]} \in \text{postfix}_{b_{i,k}}(T_{i,k})$ and since $\mathtt{W_{i,in}[X_i]}$ is the first operation in $T_{i,in}$, we have $\mathtt{W_{i,in}[X_i]} <_s \mathtt{R_{i,k}[X_i]} <_s \mathtt{C_{i,in}}$, leading to a contradiction.

We conclude that our our only remaining option is $b_{i,out} = \mathtt{R_{i,out}[X_{i+1}]}$. It now follows immediately that the next edge is indeed $(T_{i,out}, \mathtt{R_{i,out}[X_{i+1}]}, \mathtt{W_{i+1,in}[X_{i+1}]}, T_{i+1,in})$, since $\mathtt{W_{i+1,in}[X_{i+1}]}$ is the only operation conflicting with $b_{i,out}$. Furthermore, if s is allowed under READ COMMITTED, then $\mathtt{R_{i+1,in}[Z_i]}$ is in $\text{postfix}_{b_{i+1,in}}(T_{i+1,in})$, independent of our choice of $b_{i+1,in}$, as otherwise $\mathtt{W_{i,out}[Z_i]} <_s \mathtt{R_{i+1,in}[Z_i]} <_s \mathtt{C_{i,out}}$.

(C32.12) The cycle $C_{\mathcal{T}}$ consecutively contains the edges $(T_0, \mathtt{R_\emptyset[X_1]}, \mathtt{W_{1,in}[X_1]}, T_{1,in})$ and $(T_{1,in}, \mathtt{R_{1,in}[Y_1^k]}, \mathtt{W_{1,k}[Y_1^k]}, T_{1,k})$ for some $k \in \{1,2,3\}$. Furthermore, both $\text{postfix}_{\epsilon(T_{1,in})}(T_{1,in})$ and $\text{postfix}_{\epsilon(T_{1,k})}(T_{1,k})$ are nonempty.

According to Condition (C32.8), T_0 is the first transaction in $C_{\mathcal{T}}$ and $C_{\mathcal{T}}$ is transferable in T_0 on $(\mathtt{R_\emptyset[X_1]}, \mathtt{W_\emptyset[X_{n+1}]})$. The operation $\mathtt{W_{1,in}[X_1]}$ in $T_{1,in}$ is the only operation conflicting with $\mathtt{R_\emptyset[X_1]}$. Because of this, the edge $(T_0, \mathtt{R_\emptyset[X_1]}, \mathtt{W_{1,in}[X_1]}, T_{1,in})$ needs to be in $C_{\mathcal{T}}$. Since T_0 is the first transaction, the schedule s looks as follows:

$$\text{prefix}_{b_0}(T_0), \text{prefix}_{\epsilon(T_{1,in})}(T_{1,in}), \ldots, \text{postfix}_{b_0}(T_0), \text{postfix}_{\epsilon(T_{1,in})}(T_{1,in}), \ldots,$$

with $b_0 = \mathtt{R_\emptyset[X_1]}$ and $\epsilon(T_{1,in}) = \mathtt{C_{1,in}}$ or $\epsilon(T_{1,in}) = b_{1,in}$ for some operation $b_{1,in}$ in $T_{1,in}$. Note that $\mathtt{R_{1,in}[Z_\emptyset]}$ is in $\text{postfix}_{\epsilon(T_{1,in})}(T_{1,in})$, since $\mathtt{W_\emptyset[Z_\emptyset]}$ is in $\text{prefix}_{b_0}(T_0)$. Hence, $\epsilon(T_{1,in}) = b_{1,in}$ for some operation $b_{1,in}$ in $T_{1,in}$ before $\mathtt{R_{1,in}[Z_\emptyset]}$.

Analogous to our argumentation for Condition (C32.9), we can now argue that our only option for $b_{1,in}$ is $\mathtt{R_{1,in}[Y_1^k]}$ for some $k \in \{1,2,3\}$. We therefore conclude that the next edge in $C_{\mathcal{T}}$ is $(T_{1,in}, \mathtt{R_{1,in}[Y_1^k]}, \mathtt{W_{1,k}[Y_1^k]}, T_{1,k})$, and that $\text{postfix}_{\epsilon(T_{1,k})}(T_{1,k})$ is nonempty as well.

(C32.13) For every $k \in \{1,2,3\}$, if $C_{\mathcal{T}}$ contains the edge $(T_{n,k}, \mathtt{R_{n,k}[Z_n]}, \mathtt{W_{n,out}[Z_n]}, T_{n,out})$ and $\text{postfix}_{\epsilon(T_{n,out})}(T_{n,out})$ is nonempty, then the next edge in $C_{\mathcal{T}}$ is $(T_{n,out}, \mathtt{R_{n,out}[X_{n+1}]}, \mathtt{W_\emptyset[X_{n+1}]}, T_0)$.

The argumentation is analogous to Condition (C32.11). Note in particular that, since T_0 is the first transaction in $C_{\mathcal{T}}$ (Condition (C32.8)), the order of $T_{n,out}$ and T_0 is swapped in s:

$$\text{prefix}_{b_0}(T_0), \ldots, \text{prefix}_{b_{n,out}}(T_{n,out}), \text{postfix}_{b_0}(T_0), \ldots, \text{postfix}_{b_{n,out}}(T_{n,out}), \ldots,$$

with $b_0 = \mathtt{R_\emptyset[X_1]}$ and $b_{n,out} = \mathtt{R_{n,out}[X_{n+1}]}$.

The correctness of Condition (C1) and Condition (C2) now follow immediately from Conditions (C32.8)–(C32.13). □

6 SCHEDULES WITH MISSING AND REPEATING TRANSACTIONS

All the above results concern schedules in which transaction occurrences are entirely defined by set $\mathcal{T}$. That is, every transaction in $\mathcal{T}$ must occur precisely once in the schedule, and no further transactions are allowed. In this section, we explore robustness for a slightly different definition of schedules w.r.t. $\mathcal{T}$, where not every transaction must occur in the schedule and transactions can repeat.

For a formal definition, we need some additional terminology: We will say that two transactions T_i and T_j are *equivalent* if they are identical up to their associated id. For an example, the transactions $\mathtt{R_1[x]R_1[y]W_1[x]C_1}$ and $\mathtt{R_2[x]R_2[y]W_2[x]C_2}$ are equivalent. Notice that a set $\mathcal{T}$ of transactions is already allowed to contain multiple different, equivalent transactions. We say that set $\mathcal{T}$ is an *instantiation of* a another set $\mathcal{T}'$ of transactions if every transaction in $\mathcal{T}$ has an equivalent in $\mathcal{T}'$. Set $\mathcal{T}$ is *equivalent* to $\mathcal{T}'$ if, in addition, $\mathcal{T}'$ is also an instantiation of $\mathcal{T}$.

Definition 33. Let $\mathcal{T}$ be a set of transactions. A schedule s is *consistent with* $\mathcal{T}$ if it is a schedule over an instantiation of $\mathcal{T}$.

For an example, schedules $R_1[x]R_2[x]R_3[x]W_4[z]R_1[y]W_1[x]C_1R_3[y]R_2[y]W_4[x]W_2[x]C_2C_4W_3[x]C_3$ and $R_1[x]W_2[z]W_2[x]C_2R_1[y]W_1[x]C_1$ are both consistent with $\mathcal{T} = \{R_1[x]R_1[y]W_1[x]C_1, W_2[z]W_2[x]C_2\}$.

We now explore robustness under this new definition, taking every schedule into account that is consistent with (instead of *over*) a given reference set of transactions $\mathcal{T}$.

The next theorem is the main result of this section. It shows that, for all considered isolation levels, the new variant of robustness is in the same complexity class as its original formulation. Notice that this is not a straightforward result: For every non-empty set $\mathcal{T}$ of transactions there are only a finite number of schedules over $\mathcal{T}$, but there are infinitely many schedules consistent with $\mathcal{T}$.

In Theorem 34, we write $\mathcal{T} \uplus \mathcal{T}'$ to denote the (disjoint) union of $\mathcal{T}$ with an isomorphic copy of $\mathcal{T}'$ in which transaction ids are disjoint to those in $\mathcal{T}$.

Theorem 34. *Let $\mathcal{T}$ be a set of transactions. All schedules consistent with $\mathcal{T}$ allowed under a given isolation level are conflict serializable if and only if $\mathcal{T} \uplus \mathcal{T}$ is robust under the given isolation level.*

In the remainder of the section, we prove Theorem 34. For this, we first show that robustness is anti-monotone and thus that a set of transactions not robust under a given isolation level cannot be made robust under that isolation level by adding more transaction.

Lemma 35. *Let $\mathcal{T}$ be a set of transactions and T_i a transaction. Then, a schedule s is a multi-split schedule for $\mathcal{T}$ if and only if $s \cdot T_i$ is a multi-split schedule for $\mathcal{T} \uplus \{T_i\}$. Moreover, they are both based on the same transferable cycle and allowed under the same isolation levels.*

Proof. (If) By definition of multi-split schedule $s \cdot T_i$, set $\mathcal{T}$ must be non-empty and the first transaction T_1 in its transferable cycle C must be different from T_i. It then follows straightforward from Definition 18 that s is indeed a multi-split schedule for $\mathcal{T}$. Furthermore, by removing transactions from a schedule, no anomalies can be introduced.

(Only If). That $s \cdot T_i$ is a multi-split schedule for $\mathcal{T} \uplus \{T_i\}$ follows immediately from Definition 18. Since $\mathcal{T}$ is appended and closed, no anomalies can be introduced. □

Finally, we complete the proof by showing that a set of transactions not robust under a given isolation level must have a subset in which no three different equivalent transactions occur, and that by itself is already non-robust.

Proposition 36. *Let $\mathcal{T}$ be a set of transactions and $X \in$ {no isolation, read uncommitted, read committed}. The following two statements are equivalent:*

(1) *There is a multi-split schedule for $\mathcal{T}$ allowed under X; and*
(2) *There is a multi-split schedule for a subset of $\mathcal{T}$ allowed under X in which every transaction has at most one different equivalent transaction.*

Proof. The direction (2) $\Rightarrow$ (1) is straightforward, hence we focus on (1) $\Rightarrow$ (2). For this, let $\mathcal{T}$ be a set of transactions and s a multi-split schedule for $\mathcal{T}$ based on some transferable cycle C. Let T_1 be the first transaction mentioned in this cycle. From Lemma 35, it follows that removing all transactions not in C from s leads to a multi-split schedule s', still based on C, for the subset $\mathcal{T}_C \subseteq \mathcal{T}$ of transactions that are mentioned in C.

If C contains no three equivalent transactions, then the result of the proposition is immediate. Therefore, we proceed with the assumption that two different, equivalent transactions T_i and T_j exist in s' that are both different from T_1. (Notice that T_1 may be equivalent to T_i and T_j. This is why s' must contain at least three such transactions for the construction to work.) Without loss of generality, we assume that $\mathsf{prefix}_{\epsilon(T_i)}(T_i)$ occurs before $\mathsf{prefix}_{\epsilon(T_j)}(T_j)$ in s'.

In the remainder of the proof, we show that a strict subset $\mathcal{T}'$ of $\mathcal{T}$ exists, in which T_i is eliminated, but that still admits a multi-split schedule that is allowed under isolation level X. Repeating this strategy leads to the desired subset of $\mathcal{T}$ in which every transaction has at most two different equivalent transactions.

Let T_k be the transaction occurring immediately before T_i in C, thus with an edge (T_k, b_k, a_i, T_i) in C. (Notice that k may equal 1.) Then we know, due to the equivalence of T_i to T_j, that there is an edge (T_k, b_k, a'_i, T_j) in the interference graph of $\mathcal{T}_C$, with a'_i the equivalent of operation a_i in T_i for T_j. By definition of multi-split schedule, $\mathsf{prefix}_{\epsilon(T_k)}(T_k)$ occurs before $\mathsf{prefix}_{\epsilon(T_j)}(T_j)$ in s' and b_k must occur in $\mathsf{prefix}_{\epsilon(T_k)}(T_k)$. Indeed, by definition, $\epsilon(T_k)$ is either b_k or C_k.

It follows that $b_k <_{s'} a'_i$ and, as a result, that s' admits also the conflict cycle C' with edge (T_k, b_k, a'_i, T_j) and paths T_1 to T_k and T_j to T_1 as in C. It is now immediate that the schedule obtained by removing from s' all operations from transactions that are not in C', is a schedule for the set $\mathcal{T}_{C'} \subseteq \mathcal{T}_C$ of transactions in C'. Notice in particular that this schedule is allowed under the same isolation level as s' for $\mathcal{T}_C$. We have thus shown that $\mathcal{T}_{C'}$ is not robust and thus that a multi-split schedule for $\mathcal{T}_{C'} \subseteq \mathcal{T}$ exists in which T_i is no longer present. □

7 RELATED WORK

In this section, we discuss the papers that considered (variants of) the robustness problem.

Sufficient conditions. Fekete et al. [16] studied the robustness problem for SNAPSHOT ISOLATION by extending traditional conflict graphs with extra information w.r.t. the type of each conflict. In contrast to our interference graphs, these static dependency graphs only capture the possible types of conflicts between transactions but not the specific operations responsible for these conflicts. Based on these graphs, a sufficient condition for robustness against SNAPSHOT ISOLATION is presented, as well as possible approaches on how to modify transactions when robustness is not guaranteed. The performance of these approaches is studied by Alomari et al. [2]. Alomari and Fekete [3] provide a sufficient condition for robustness against READ COMMITTED, both under a lock-based and multiversion semantics. This work uses the same graph approach as in Reference [16]. The provided condition, however, is not a necessary condition and can therefore not be used to characterize robustness against READ COMMITTED.

Cerone et al. [11] provide a framework for uniformly specifying different isolation levels in a declarative way. A key assumption in their framework is *atomic visibility*, requiring that either all or none of the updates of each transaction are visible to other transactions. This assumption facilitates reasoning over isolation levels, since these isolation levels can be specified by consistency axioms on the level of transactions instead of individual operations within each transaction. Bernardi and Gotsman [10] extended the work of Fekete et al. [16] by providing sufficient conditions for robustness against the different isolation levels that can be defined by this framework. Continuing on this line of work, Cerone, Gotsman, and Yang [13] examined the relationship between consistency axioms restricting the allowed schedules over a set of transactions and the resulting properties of possible cycles in the static dependency graph for this set of transactions. In particular, they provide a more direct approach to derive robustness criteria based on static dependency graphs from arbitrary isolation levels specified by consistency axioms. Cerone and Gotsman [12] later refined the sufficient condition for robustness against SNAPSHOT ISOLATION first obtained by Fekete et al. [16]. They furthermore obtained a sufficient condition for robustness against PARALLEL SNAPSHOT ISOLATION toward SNAPSHOT ISOLATION (i.e., whether for a given workload every schedule allowed under PARALLEL SNAPSHOT ISOLATION is allowed under snapshot isolation). However, the declarative framework by Cerone et al. [11] providing the foundation on which the above work is built, cannot be used to study read committed (and hence read uncommitted) as it does not admit the atomic visibility condition.

Characterizations. As mentioned before, Fekete [15] is the first work that provides a necessary and sufficient condition for deciding robustness against snapshot isolation. In particular, that work provides a characterization for acceptable allocations when every transaction runs under either snapshot isolation or strict two-phase locking. The allocation then is acceptable when every possible execution respecting the allocated isolation levels is serializable. As a side result, this work indirectly provides a necessary and sufficient condition for robustness against snapshot isolation, since robustness against snapshot isolation holds iff the allocation where each transaction is allocated to snapshot isolation is acceptable.

Beillahi et al. use an algorithmic approach to decide robustness against causal consistency [8] and snapshot isolation [7] by providing a polynomial time reduction from these problems to the reachability problem in transactional programs over a sequentially consistent shared memory. Their setting is slightly different from our setting, as they allow a nondeterministic execution of transactions. They furthermore group transactions under different processes. During execution, each process then runs its transactions sequentially but concurrently with other processes. Due to this different setting, they obtain complexity bounds that are considerably higher than our complexity results. In particular, they show that deciding robustness against causal consistency and snapshot isolation are expspace-complete in general, and pspace-complete if respectively the number of sites or the number of processes is fixed.

Transaction chopping. Instead of weakening the isolation level, transactions can also be split in smaller pieces to obtain performance benefits. However, this approach poses a new challenge, as not every serializable execution of these chopped transactions is necessarily equivalent to some serializable execution over the original transactions. A chopping of a set of transactions is correct if for every serializable execution of the chopping there exists an equivalent serializable execution of the original transactions. Shasha et al. [20] provide a graph-based characterization for this correctness problem. It is interesting to note that robustness against no isolation corresponds to the correctness of fully chopped transactions. Indeed, if we chop each transaction into pieces consisting of single operations, then every serializable schedule of this chopping would clearly correspond to a schedule over the original transactions allowed under no isolation and vice versa. However, this relation is no longer trivial when considering robustness against read uncommitted and read committed. In particular, a correspondence between transaction chopping correctness and robustness against read committed is not to be expected, as the former is decidable in polynomial time [20], whereas we showed that the latter to be conp-complete.

8 CONCLUSIONS

In this article, we provided characterizations for robustness against the isolation levels read uncommitted and read committed and used these to establish upper bounds on the complexity of the associated decision problem. We also obtained matching lower bounds. The obtained characterizations provide insight into what robustness means in these settings and under which circumstances it can occur.

While the characterizations in this article are not restricted to the traditional lock-based semantics of the SQL isolation levels as they are defined in terms of forbidden patterns [9], it would be interesting to see what kind of characterizations for robustness can be found in terms of a multiversion definition of the isolation levels [1]. We provide a characterization for robustness against multiversion read committed in Reference [22]. Surprisingly, robustness against multiversion

READ COMMITTED is decidable in polynomial time, which should be contrasted with the CONP-hardness for robustness against READ COMMITTED obtained in this article. A second immediate question pertains the CONP-hardness result: Are there natural restrictions that make the problem tractable? In an online context with millions of transactions, testing robustness against READ COMMITTED is obviously not feasible, and tractable restrictions or approximations would be desirable. However, in an offline context, where the set of transactions is generated through a finite (and small) set of transaction programs, as discussed next, intractability is not necessarily problematic.

The initial motivation for the study of robustness lies in the performance improvement gained by executing transactions at a weaker isolation level without the danger of introducing anomalies [16]. It is important to point out that robustness makes the most sense in settings where transactions can be grouped together or where the set of possible transactions is known beforehand. A natural occurrence of the latter is when transactions are generated by a finite set of parameterized transaction programs as, for example, in a banking application where customers can do a fixed number of financial transactions. Consider the parameterized transaction $\tau = \mathtt{R}[v]\mathtt{W}[v]\mathtt{R}[w]\mathtt{R}[w]$ that represents a transfer from an account v to an account w and where v and w are variables. Any transactions $T = \mathtt{R}[\mathtt{x}]\mathtt{W}[\mathtt{x}]\mathtt{R}[\mathtt{y}]\mathtt{R}[\mathtt{y}]$ with $\mathtt{x}, \mathtt{y} \in \mathbf{Obj}$ then is an instance of τ. For this example, it could even make sense to interpret v and w with the same object $\mathtt{x}$. However, in some scenarios it makes sense to disallow different variables to be interpreted by the same object. In Reference [22], we study the robustness problem w.r.t. a formalization of parameterized transactions. In such a setting the same characterizations continue to hold but the interference graphs become infinitely large. This formalization is further extended in Reference [23] by including functional constraints, which are useful for capturing data dependencies like foreign keys.

Robustness is a binary property: A set of transactions is robust against a given isolation level or it is not. When robustness does not hold, one can devise methods to make a set of transactions robust or one can split up transactions into maximally robust subsets. These questions have been considered for SNAPSHOT ISOLATION [12, 16] and it would make sense to consider them w.r.t. the different isolation levels occurring in database systems [5]. An orthogonal, and undoubtedly more challenging, setting, is to depart from the requirement that every transaction has to be executed at the same isolation level. That is, for a given set of transaction programs, allocate every transaction to the optimal isolation level for suitable notions of optimality. An immediate interpretation of optimality could be the weakest possible isolation level for every transaction that guarantees overall robustness for the whole set. Fekete [15] studied, and solved, the allocation problem w.r.t. SNAPSHOT ISOLATION and strict two-phase locking, but no results of this flavor have been obtained for other isolation levels.

REFERENCES

[1] Atul Adya, Barbara Liskov, and Patrick E. O'Neil. 2000. Generalized isolation level definitions. In *ICDE*. 67–78.
[2] Mohammad Alomari, Michael Cahill, Alan Fekete, and Uwe Rohm. 2008. The cost of serializability on platforms that use snapshot isolation. In *ICDE*. 576–585.
[3] Mohammad Alomari and Alan Fekete. 2015. Serializable use of read committed isolation level. In *AICCSA*. 1–8.
[4] Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity—Modern Approach*. Cambridge University Press.
[5] Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. 2013. Highly available transactions: Virtues and limitations. *Proc. VLDB* 7, 3 (2013), 181–192. https://doi.org/10.14778/2732232.2732237
[6] Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. 2013. HAT, not CAP: Towards highly available transactions. In *USENIX HotOS*. 24–24.
[7] Sidi Mohamed Beillahi, Ahmed Bouajjani, and Constantin Enea. 2019. Checking robustness against snapshot isolation. In *CAV*. 286–304.
[8] Sidi Mohamed Beillahi, Ahmed Bouajjani, and Constantin Enea. 2021. Robustness against transactional causal consistency. *Log. Methods Comput. Sci.* 17, 1 (2021).

[9] 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 *SIGMOD*. 1–10.

[10] Giovanni Bernardi and Alexey Gotsman. 2016. Robustness against consistency models with atomic visibility. In *CONCUR*. 7:1–7:15.

[11] Andrea Cerone, Giovanni Bernardi, and Alexey Gotsman. 2015. A framework for transactional consistency models with atomic visibility. In *CONCUR*. 58–71.

[12] Andrea Cerone and Alexey Gotsman. 2018. Analysing snapshot isolation. *J. ACM* 65, 2 (2018), 1–41.

[13] Andrea Cerone, Alexey Gotsman, and Hongseok Yang. 2017. Algebraic laws for weak consistency. In *CONCUR*. 26:1–26:18.

[14] Stephen A. Cook and Pierre McKenzie. 1987. Problems complete for deterministic logarithmic space. *J. Algor.* 8, 3 (1987), 385–394. https://doi.org/10.1016/0196-6774(87)90018-6

[15] Alan Fekete. 2005. Allocating isolation levels to transactions. In *PODS*. 206–215.

[16] Alan Fekete, Dimitrios Liarokapis, Elizabeth J. O'Neil, Patrick E. O'Neil, and Dennis E. Shasha. 2005. Making snapshot isolation serializable. *ACM Trans. Datab. Syst.* 30, 2 (2005), 492–528. https://doi.org/10.1145/1071610.1071615

[17] Bas Ketsman, Christoph Koch, Frank Neven, and Brecht Vandevoort. 2020. Deciding robustness for lower SQL isolation levels. In *PODS*. ACM, 315–330.

[18] Christos H. Papadimitriou. 1986. *The Theory of Database Concurrency Control*. Computer Science Press.

[19] Omer Reingold. 2008. Undirected connectivity in log-space. *J. ACM* 55, 4 (2008), 17:1–17:24. https://doi.org/10.1145/1391289.1391291

[20] Dennis E. Shasha, François Llirbat, Eric Simon, and Patrick Valduriez. 1995. Transaction chopping: Algorithms and performance studies. *ACM Trans. Datab. Syst.* 20, 3 (1995), 325–363. https://doi.org/10.1145/211414.211427

[21] TPC-C. On-Line Transaction Processing Benchmark. Retrieved from http://www.tpc.org/tpcc/.

[22] Brecht Vandevoort, Bas Ketsman, Christoph Koch, and Frank Neven. 2021. Robustness against read committed for transaction templates. *Proc. VLDB Endow.* 14, 11 (2021), 2141–2153. https://doi.org/10.14778/3476249.3476268

[23] Brecht Vandevoort, Bas Ketsman, Christoph Koch, and Frank Neven. 2022. Robustness against read committed for transaction templates with functional constraints. In *ICDT*. 16:1–16:17. https://doi.org/10.4230/LIPIcs.ICDT.2022.16

[24] Gerhard Weikum and Gottfried Vossen. 2002. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann.

Received 16 November 2020; revised 27 January 2022; accepted 24 August 2022