Robustness against Consistency Models with Atomic Visibility

Giovanni Bernardi and Alexey Gotsman

IMDEA Software Institute, Madrid, Spain

Abstract

To achieve scalability, modern Internet services often rely on distributed databases with consistency models for transactions weaker than serializability. At present, application programmers often lack techniques to ensure that the weakness of these consistency models does not violate application correctness. We present criteria to check whether applications that rely on a database providing only weak consistency are *robust*, i.e., behave as if they used a database providing serializability. When this is the case, the application programmer can reap the scalability benefits of weak consistency while being able to easily check the desired correctness properties. Our results handle systematically and uniformly several recently proposed weak consistency models, as well as a mechanism for strengthening consistency in parts of an application.

1998 ACM Subject Classification C.2.4 Distributed Systems

Keywords and phrases Robustness, Replication, Consistency models, Transactions

1 Introduction

To achieve scalability and availability, modern Internet services often rely on large-scale databases that replicate and partition data across a large number of nodes and/or a wide geographical span (e.g., [3, 4, 5, 6, 9, 10, 13, 19, 21, 25, 26]). The database clients invoke transactions on the data at any of the nodes, and the nodes communicate changes to each other using message passing. Ideally, we want this distributed system to provide strong guarantees about transaction processing, such as *serializability* [7]: the results of concurrently executing a set of transactions could be obtained if these transactions were executed serially in some order. Serializability is useful because it allows an application programmer to easily establish desired correctness properties. For example, to check that the transactions of an application preserve a given data integrity constraint, the programmer only needs to check that every transaction does so when executed in isolation, without worrying about concurrency. Unfortunately, achieving serializability requires excessive synchronisation among database nodes, which slows down the database and even makes it unavailable if network connections between replicas fail [1, 16]. For this reason, nowadays large-scale databases often provide weak consistency guarantees, which allow non-serializable behaviours, called *anomalies*.

As a motivating example, consider a toy on-line auction application with transactions defined by the *transactional programs* in Figure 1. The program `RegUser` creates a new user account. It manipulates the table USERS, whose rows contain a primary key (uId) and a nickname. An invocation of `RegUser(uname)` inserts a new row in USERS only if the nickname `uname` does not appear in USERS, to ensure that nicknames are unique. The program `ViewUsers` can be used to view all the users. Some databases [3, 21, 25] may allow executions of `RegUser` and `ViewUsers` such as the one sketched in Figure 2(c). There two invocations of `RegUser` generate the transactions T_1 and T_2; these write two rows of USERS, denoted by x and y, to register the users *Alice* and *Bob*. The program `ViewUsers()` then is

27th International Conference on Concurrency Theory (CONCUR 2016).
Editors: Josée Desharnais and Radha Jagadeesan; Article No. 07; pp. 07:1–07:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

```
RegUser(uname)
select name
from USERS
as ret_name
where name = uname;
if ret_name defined
then abort
else
insert into USERS
values (new uId, uname);
```

```
StoreBid(iid,val)
insert into BIDS
values
(new bId, iid, val);
select nbids
from ITEMS
as n where iId = iid;
update ITEMS
set nbids = n + 1
where iId = iid;
```

```
ViewItem(iid)
select *
from ITEMS as ret_item
where iId = iid;
```

```
ViewUsers()
select *
from USERS
as ret_users;
```

USERS(uId, name)
ITEMS(iId, desc, nbids)
BIDS(bId, iId, val)

Figure 1 SQL code and table schema for an auction application. Primary keys are underlined.

invoked twice; the invocation in T_3 sees *Alice* but not *Bob*, while the invocation in T_4 sees *Bob* but not *Alice*. This result, called a *long fork* anomaly, cannot be obtained by executing the four transactions in any sequence and, hence, is not serializable.

The past few years have seen a number of proposals of new transactional consistency models for modern large-scale databases [3, 5, 10, 21, 25], differing in how much they weaken consistency, by exposing such anomalies, in exchange for improved performance. Unfortunately, application programmers often lack techniques to ensure that the weakness of these consistency models does not violate application correctness. This situation hinders the adoption of the novel consistency models by mainstream database developers and application programmers.

One way to address this problem is using the notion of application *robustness* [8, 14, 15]. An application is robust against a particular weak consistency model if it behaves the same whether using a database providing this model or serializability. If an application is robust against a given weak consistency model, then programmers can reap the performance benefits of using weak consistency while being able to easily check the desired correctness properties.

In this paper we develop criteria for checking the robustness of applications against three recently proposed consistency models—causal (aka causal+) consistency (CC) [21], prefix consistency [10] (PC) and parallel snapshot isolation [25] (PSI, aka non-monotonic snapshot isolation [3]). As a corollary of our results, we also derive an existing robustness criterion [15] for a classical model of snapshot isolation [6] (SI). Our criteria also handle variants of the consistency models that allow application programmers to request that certain transactions be executed under serializability and thereby ensure the robustness of applications that are not robust otherwise.

We handle the above four consistency models in a uniform and systematic way by exploiting a recently proposed framework [11] for declaratively specifying their semantics (§2). In particular, all of the consistency models that we consider guarantee the *atomic visibility* of transactions: either all or none of the writes performed by a transaction can be observed by other transactions. This allows us to simplify reasoning needed to establish robustness criteria by abstracting from internals of transactions in application executions. We first propose a *dynamic* robustness criterion that checks whether a given execution is serializable (§3). We formulate this criterion in terms of the dependency graph of the execution [2], describing several kinds of relationships between its transactions: an execution is serializable if its dependency graph contains no cycles of a certain form, which we call *critical*. Criteria for robustness against different consistency models differ in which cycles are considered critical. We then illustrate how our dynamic robustness criteria on a single execution can be lifted to *static* criteria that check that all executions of a given application are serializable (§4).

2 Consistency Model Specifications

We start by recalling from [11] a formal model of database computations and the specifications of the consistency models that we handle. These specifications are declarative, which greatly simplifies our formal development. Nonetheless, as shown in [11], the specifications are equivalent to certain operational specifications, close to implementations.

We consider a database storing ***objects*** $\mathsf{Obj} = \{x, y, \ldots\}$, which we assume to be natural-valued. Clients interact with the database by issuing `read` and `write` operations on the objects, grouped into transactions. We denote each operation invocation by an ***event*** (ι, o), where ι is an identifier from a denumerable set $\mathsf{EventId}$, and $o \in \{\mathtt{read}(x, n), \mathtt{write}(x, n) \mid x \in \mathsf{Obj}, n \in \mathbb{N}\}$ describes the operation invoked and its outcome: reading a value n from an object x or writing n to x. We range over events by e, f, g and denote the set of all events by Event. In the following we denote irrelevant expressions by _, and write $e \vdash \mathtt{write}(x, n)$ if $e = (\_, \mathtt{write}(x, n))$ and $e \vdash \mathtt{read}(x, n)$ if $e = (\_, \mathtt{read}(x, n))$. A binary relation $<$ is a ***strict partial order*** if it is transitive and irreflexive. It is ***total*** if additionally for all elements a and b, we have $a < b$, $b < a$ or $a = b$.

▶ **Definition 1.** A ***transaction*** $T, S, \ldots$ is a pair $(E, <_{\mathsf{po}})$, where $E \subseteq \mathsf{Event}$ is a finite, non-empty set of events with distinct identifiers, and the ***program order*** $<_{\mathsf{po}}$ is a total order over E. A ***history*** H is a finite set of transactions with disjoint sets of event identifiers. An annotated history (H, level) is a pair where H is an history and $\mathsf{level} : H \to \{\mathsf{SER}, \bot\}$. An ***execution*** is a triple $X = ((H, \mathsf{level}), <_{\mathsf{hb}}, <_{\mathsf{ar}})$, where (H, level) is an annotated history, $<_{\mathsf{hb}}$ is a strict partial order over H, and $<_{\mathsf{ar}}$ is a total order over H such that $<_{\mathsf{hb}} \subseteq <_{\mathsf{ar}}$. We refer to $<_{\mathsf{hb}}$ and $<_{\mathsf{ar}}$ as ***happens-before*** and ***arbitration***. ◀

We denote components of an execution as in $X.H$ and use the same notation for similar structures.

A transaction records a set of operations and the order in which the client program invoked them. A history records transactions that committed in a finite database computation. For simplicity we elide the treatment of aborted and ongoing transactions, as well as infinite database computations. Annotated histories enrich histories with a function level that records which transactions the programmer requested to execute under serializability, and which transactions under the weak consistency model offered by the underlying database. Finally, executions enrich annotated histories with a happens-before order and an arbitration order, which declaratively represent internal database processing. Intuitively, $T <_{\mathsf{hb}} S$ means that S is aware of the updates performed by T, and thus the outcome of the operations in S may depend on the effects of T. We call transactions that are not related by happens-before ***concurrent***. The relationship $T <_{\mathsf{ar}} S$ means that the versions of objects written by S supersede those written by T. The constraint $<_{\mathsf{hb}} \subseteq <_{\mathsf{ar}}$ ensures that writes by a transaction T supersede those that T is aware of.

We use the set $\{\mathsf{CC}, \mathsf{PC}, \mathsf{PSI}, \mathsf{SI}, \mathsf{SER}\}$ to refer to the consistency models that we treat (§1), and we range over this set by wm. In Figure 4 we specify these consistency models as combination of the axioms in Figure 3, constraining executions. Formally, we let the set of annotated histories allowed by a consistency model wm be given by $\mathsf{hist}(wm) = \{(X.H, X.\mathsf{level}) \mid X \models wm\}$. We now explain the axioms and the anomalies that they (dis)allow. We summarise these anomalies in Figure 2.

Given a total order $< \subseteq A \times A$ and a set $B \subseteq A$, we write $\max(B, <)$ for the element $b \in B$ such that $\forall a \in B.\, a \leq b$; if $A = \emptyset$, then $\max(B, <)$ is undefined. We define min in the

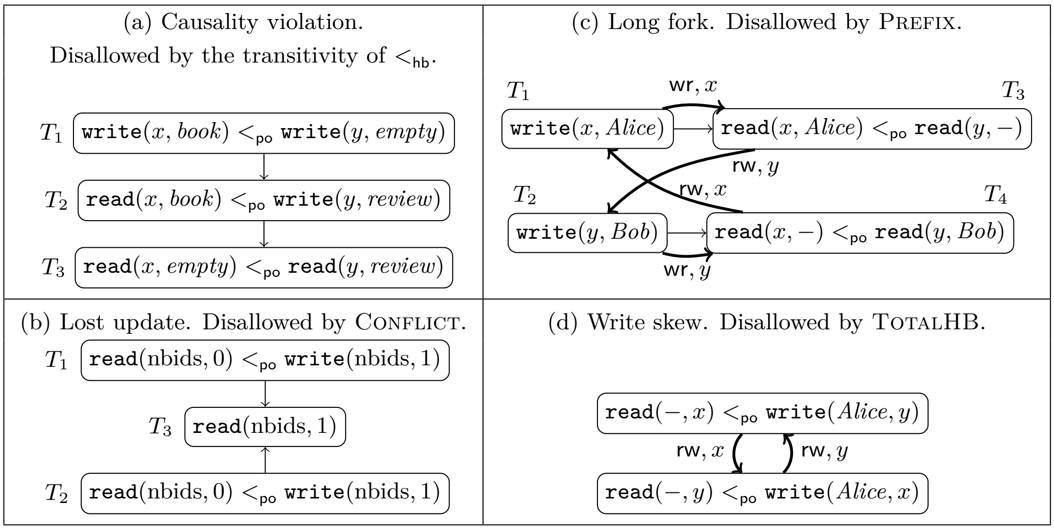


Figure 2 Non-serializable executions illustrating anomalies. Boxes represent transactions, and thin arrows between boxes represent the happens-before relation. We omit arbitration edges to avoid clutter. The thick arrows marked wr/rw are explained in §3.

$\forall (E, <_{\mathsf{po}}) \in H.\, \forall e \in E.\, \forall x, n.$ $e \vdash \mathtt{read}(x, n) \implies (\mathsf{before}(e, <_{\mathsf{po}}, \_\ x) = \emptyset \lor \max(\mathsf{before}(e, <_{\mathsf{po}}, \_\ x), <_{\mathsf{po}}) \vdash \_(\_, n))$	(INT)
$\forall T \in H.\, \forall x, n.$ $T \vdash \mathtt{read}(x, n) \implies ((\mathsf{before}(T, <_{\mathsf{hb}}, \mathtt{write}\ x) = \emptyset \land n = 0) \lor$ $\max(\mathsf{before}(T, <_{\mathsf{hb}}, \mathtt{write}\ x), <_{\mathsf{ar}}) \vdash \_(\_, n))$	(EXT)
$\forall T, S \in H.\, (\exists x.\, T \vdash \mathtt{write}(x, \_) \land S \vdash \mathtt{write}(x, \_)) \implies T = S \lor T <_{\mathsf{hb}} S \lor S <_{\mathsf{hb}} T$	(CONFLICT)
$<_{\mathsf{ar}} ; <_{\mathsf{hb}} \subseteq <_{\mathsf{hb}}$	(PREFIX)
$\forall T, S \in H.\, T = S \lor T <_{\mathsf{hb}} S \lor S <_{\mathsf{hb}} T$	(TOTALHB)
$\forall T, S \in H.\, (\mathsf{level}(T) = \mathsf{level}(S) = \mathsf{SER}) \implies T = S \lor T <_{\mathsf{hb}} S \lor S <_{\mathsf{hb}} T$	(SERTOTAL)

Figure 3 Consistency axioms constraining an execution $((H, \mathsf{level}), <_{\mathsf{hb}}, <_{\mathsf{ar}})$.

obvious dual manner. In the following, when we write $\max(B, <)$ or $\min(B, <)$, we assume that they are defined. Given a partial order $< \subseteq A \times A$ and an $a \in A$, we define the downset of a as $\mathsf{before}(a, <) = \{a' \in A \mid a' < a\}$, and let $\mathsf{before}(a, <, \mathsf{op}\ x) = \mathsf{before}(a, <) \cap \{a' \in A \mid a' \vdash \mathsf{op}(x, \_)\}$.

The ***internal consistency axiom*** INT ensures that, within a transaction, the database provides sequential semantics: in a transaction $(E, <_{\mathsf{po}})$, a read event e on an object x returns the value of the last event on x preceding e. The events on x preceding e are given by the set $\mathsf{before}(e, <_{\mathsf{po}}, \_\ x)$. If in $(E, <_{\mathsf{po}})$ a read e on x is not preceded by an operation on the same object (i.e., $\mathsf{before}(e, <_{\mathsf{po}}, \_\ x) = \emptyset$), then its value is determined in terms of writes by other transactions, using the ***external consistency axiom*** EXT. To formulate EXT we lift the $\vdash$ notation to transactions. For given $T = (E, <_{\mathsf{po}})$, $x \in \mathsf{Obj}$ and $n \in \mathbb{N}$, we write:

- $T \vdash \mathtt{write}(x, n)$ if $\max(\{e \in E \mid e \vdash \mathtt{write}(x, \_)\}, <_{\mathsf{po}}) \vdash \mathtt{write}(x, n)$; and
- $T \vdash \mathtt{read}(x, n)$ if $\min(\{e \in E \mid e \vdash \_(x, \_)\}, <_{\mathsf{po}}) \vdash \mathtt{read}(x, n)$.

$\mathsf{CC} \equiv \textsc{Int} \wedge \textsc{Ext} \wedge \textsc{SerTotal}$
$\mathsf{PSI} \equiv \mathsf{CC} \wedge \textsc{Conflict}$
$\mathsf{PC} \equiv \mathsf{CC} \wedge \textsc{Prefix}$
$\mathsf{SI} \equiv \mathsf{PSI} \wedge \textsc{Prefix}$
$\mathsf{SER} \equiv \textsc{Int} \wedge \textsc{Ext} \wedge \textsc{TotalHB}$

Figure 4 Consistency model definitions.

According to Ext, if a transaction T reads an object x before writing to it, then the value returned by the read is determined by the transactions that happen before T and that write to x; the set of such transactions is given by $\mathsf{before}(T, <_{\mathsf{hb}}, \mathtt{write}\ x)$. If this set is empty, then T reads the initial value 0; otherwise it reads the value written by the transaction from the set that is the last one in $<_{\mathsf{ar}}$. Ext guarantees the *atomic visibility* of a transaction: either all or none of its writes can be visible to another transaction. A detailed discussion on the matter can be found in [11, Section 3].

The axiom SerTotal formalises the additional guarantees provided to transactions that the application programmer required to execute on serializability, as recorded by level. We discuss this axiom in more detail below; for now we assume $\mathsf{level} = (\lambda T.\, \bot)$ for all executions.

The axioms Int, Ext and SerTotal define causal consistency (CC) [21]. This forbids the ***causality violation*** anomaly in Figure 2(a), where a user sees the review, but not the book it was associated with. This anomaly is forbidden because $<_{\mathsf{hb}}$ is transitive, so we must have $T_1 <_{\mathsf{hb}} T_3$. Since $T_1 <_{\mathsf{ar}} T_2$, the writes by T_2 supersede those by T_1, and thus Ext implies $T_3 \vdash \mathtt{read}(y, \mathit{review})$ and $T_3 \vdash \mathtt{read}(x, \mathit{book})$.

Causal consistency allows the ***lost update*** anomaly, illustrated by the execution in Figure 2(b). This execution may arise from the programs `ViewItem` and `StoreBid` in Figure 1, which respectively let a user query the information about an item and bid on an item. They access a table ITEMS, whose rows represent items and contain a primary key (iId), an item description (desc) and the number of existing bids (nbids). The anomaly in Figure 2(b) is caused by two invocations of the program `StoreBid` that generate the transactions T_1 and T_2, meant to increase the number of bids of an item. The two transactions read the initial number of bids for the item, namely 0, and concurrently modify it, resulting in one addition getting lost. This is observed by a third transaction T_3 generated by `ViewItem`. The lost update anomaly is disallowed by the axiom Conflict, which guarantees that transactions updating the same object are not concurrent. This axiom rules out any execution with the history in Figure 2(b). We specify parallel snapshot isolation (PSI) [25] by strengthening causal consistency with the axiom Conflict. This consistency models allows the ***long fork*** anomaly given in Figure 2(c), which we discussed in §1.

We specify prefix consistency (PC) [10] and snapshot isolation (SI) [6] by strengthening respectively CC and PSI via the axiom Prefix: if T observes S, then it also observes all $<_{\mathsf{ar}}$-predecessors of S, which is formalised using sequential composition ; of relations. The axiom Prefix disallows any execution with the history in Figure 2(c): T_1 and T_2 have to be related by $<_{\mathsf{ar}}$ one way or another; but then by Prefix, either T_4 has to observe *Alice* or T_3 has to observe *Bob*. Like causal consistency, prefix consistency allows the lost update anomaly in Figure 2(b). Snapshot isolation disallows it, but allows the anomaly of ***write skew***, illustrated by the execution in Figure 2(d). This execution could be produced by `RegUser` in Figure 1. The objects x and y correspond to different rows in the table USERS. Two invocations of `RegUser` generate transactions that miss each other's writes and, as a consequence, concurrently register two users with the same nickname.

We define serializability (SER) using the axiom TotalHB, which requires happens-before to be total. It disallows any execution with one of the histories in Figure 2.

Finally, the consistency models we consider include the axiom SerTotal, which requires happens-before to be total on transactions that the programmer marked as serializable.

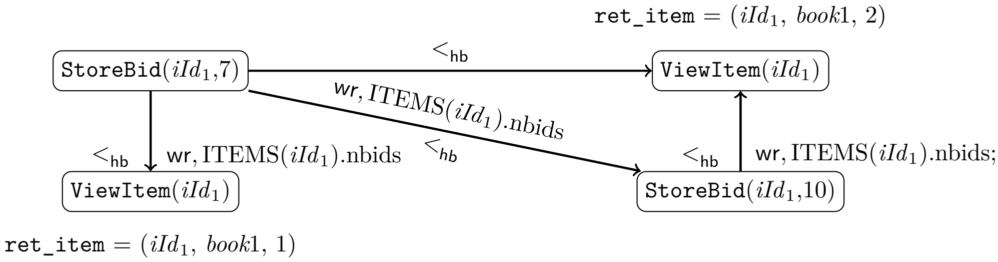


Figure 5 An execution produced by the programs in Figure 1 and its dynamic dependency graph (the latter explained in §3). We assume $\mathsf{level} = (\lambda T.\, \bot)$. We omit events inside transactions and only show the parameters and the return values of the corresponding programs. Since $<_{\mathsf{hb}} \subseteq <_{\mathsf{ar}}$, all the relevant arbitration edges coincide with the happens-before ones, and we omit the $<_{\mathsf{ar}}$ label.

For example, in a database providing CC, the history in Figure 2(c) can be disallowed by letting $\mathsf{level}(T_1) = \mathsf{level}(T_2) = \mathsf{SER}$ and $\mathsf{level}(T_3) = \mathsf{level}(T_4) = \bot$. This is because then SerTotal forces T_1 and T_2 to be related by happens-before, and therefore either T_3 or T_4 has to observe *both* T_1 and T_2. We can disallow the history in Figure 2(b) by letting $\mathsf{level}(T_1) = \mathsf{level}(T_2) = \mathsf{SER}$ and $\mathsf{level}(T_3) = \bot$.

As the last example, we consider the execution X in Figure 5, which is produced on a PSI database by the programs `StoreBid` and `ViewItem` in Figure 1: $X \models \mathsf{PSI}$. In X the two transactions due to `StoreBid` submit bids for an item iId_1: one bid of 7 dollars and one bid of 10 dollars. The other two transactions due to `ViewItem` query the state of the item. The query on the left sees the bid of 7, but not that of 10. The query on the right sees both bids. It is easy to check that the history of this execution is serializable. As a matter of fact, the results we develop in the following sections let us show that *any* execution produced by the programs `StoreBid` and `ViewItem` under PSI has a serializable history. Hence, a database can process the corresponding transactions using the PSI concurrency control without exposing any anomalies to its users.

3 Dynamic Robustness Criteria

Our first goal is to define criteria to check whether a single execution X in one of the consistency models that we consider has a serializable history: $X.H \in \mathsf{hist}(\mathsf{SER})$. From these *dynamic* robustness criteria, in the next section we derive *static* criteria to check whether this is the case for all executions of a given application.

Our dynamic criteria are formulated in terms of *dependency graphs*, widely used in the database literature [2]. Let the set of ***labels*** L be defined as follows: $D = \{(\mathsf{wr}, x), (\mathsf{ww}, x) \mid x \in \mathsf{Obj}\}$, $L = D \cup \{(\mathsf{rw}, x) \mid x \in \mathsf{Obj}\}$. We use λ to range over L and s, t to range over $L^\star$. A graph G is a pair $(H, \longrightarrow)$, where $H \in \mathsf{Hist}$ and $\longrightarrow \subseteq H \times L \times H$. We write $T \xrightarrow{\lambda} S \in G$ in place of $(T, \lambda, S) \in \longrightarrow$. We also use some graph-theoretic notions. A ***path*** π in G is a non-empty finite sequence of edges $T_0 \xrightarrow{\lambda_0} T_1 \xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_{n-1}} T_n$. In this case we write $\pi \in G$. The path is a ***cycle*** if $T_0 = T_n$, and it is a ***simple*** cycle if all other pairs of transactions on it are distinct. We also write $T \xrightarrow{\lambda} S \in \pi$ to mean that the edge $T \xrightarrow{\lambda} S$ appears in the path π. Given a graph $G = (H, \longrightarrow)$, we denote by $\xrightarrow{s}^{+}$ the least relation such that (i) $T \xrightarrow{\lambda}^{+} S$ if $T \xrightarrow{\lambda} S \in G$; and (ii) $T \xrightarrow{\lambda s}^{+} S$ whenever $T \xrightarrow{\lambda} T' \in G$ and $T' \xrightarrow{s}^{+} S$ for some $T' \in H$. We denote with $\xrightarrow{s}^{\star}$ the reflexive closure of $\xrightarrow{s}^{+}$, so that $T \xrightarrow{\varepsilon}^{\star} T$ for every $T \in H$. We now define a map from executions into dependency graphs.

▶ **Definition 2.** The ***dynamic dependency graph*** of an execution $X = ((H, \_), <_{\mathsf{hb}}, <_{\mathsf{ar}})$ is $\mathsf{DDG}(X) = (H, \longrightarrow)$, where for every $x \in \mathsf{Obj}$ the relation $\longrightarrow$ contains the following triples:

read-dependency: $T \xrightarrow{\mathsf{wr},\, x} S$ if $S \vdash \mathtt{read}(x, \_)$ and $T = \max(\mathsf{before}(S, <_{\mathsf{hb}}, \mathtt{write}\ x), <_{\mathsf{ar}})$;

write-dependency: $T \xrightarrow{\mathsf{ww},\, x} S$ if $T \vdash \mathtt{write}(x, \_)$, $S \vdash \mathtt{write}(x, \_)$ and $T <_{\mathsf{ar}} S$;

anti-dependency: $T \xrightarrow{\mathsf{rw},\, x} S$ if $T \neq S$ and either (i) $T \vdash \mathtt{read}(x, \_)$, $S \vdash \mathtt{write}(x, \_)$ and $\mathsf{before}(T, <_{\mathsf{hb}}, \mathtt{write}\ x) = \emptyset$, or (ii) $T' \xrightarrow{\mathsf{wr},\, x} T$ and $T' \xrightarrow{\mathsf{ww},\, x} S$ for some $T' \in H$. ◀

Thus, $T \xrightarrow{\mathsf{wr},\, x} S$ means that S reads T's write to x (cf. Ext in Figure 3), and $T \xrightarrow{\mathsf{ww},\, x} S$ means that S overwrites T's write to x. The relation $T \xrightarrow{\mathsf{rw},\, x} S$ means that S overwrites the write to x read by T (the initial value of an object is overwritten by any write to this object). In Figures 2(c), 2(d) and 5 we draw the dependency graphs with thick edges.

Dependency graphs provide a way to show that executions have serializable histories [2].

▶ **Lemma 3.** *For every X, if $X \models \text{Int} \wedge \text{Ext}$ and $\mathsf{DDG}(X)$ is acyclic, then $X.H \in \mathsf{hist}(\mathsf{SER})$.*

For instance, the history in Figure 5 is serializable. The graphs of the executions in Figure 2(c, d) contain cycles and, in fact, the histories of these executions are not serializable.

As we now show, to ensure that the history of an execution X arising from a particular consistency model is serializable, it is enough to check that $\mathsf{DDG}(X)$ does not contain cycles of a particular form, which we call *critical*. This more precise characterisation is instrumental in obtaining our static robustness criteria (§4).

A path π in a dynamic dependency graph G is ***chord-free*** if, whenever $u \xrightarrow{s}^{+} v \in \pi$ for some $s \in L^n$ with $n \geq 2$, we have $\neg(u \xrightarrow{\_} v \in G)$. A path π is rw-***minimal*** if, whenever $u \xrightarrow{\mathsf{rw},\, \_} v \in \pi$ and $u \xrightarrow{\lambda} v \in G$, we have $\lambda = (\mathsf{rw}, \_)$. The last notion forces us exclude an rw edge from π if there is another option.

▶ **Definition 4.** Given an execution X, an edge $T \xrightarrow{\lambda} S \in \mathsf{DDG}(X)$ is ***unprotected*** if either $X.\mathsf{level}(T) \neq \mathsf{SER}$ or $X.\mathsf{level}(S) \neq \mathsf{SER}$. A cycle $\pi \in \mathsf{DDG}(X)$ among transactions $T_0, T_1, \ldots, T_n$ (where $T_0 = T_n$) that is simple, chord-free and rw-minimal is:

CC-**critical,** if π contains an unprotected edge $T_i \xrightarrow{\mathsf{rw},\, \_} T_{i+1}$ and an unprotected edge $T_j \xrightarrow{\lambda} T_{j+1}$ with $i \neq j$ and $\lambda \in \{(\mathsf{ww}, \_), (\mathsf{rw}, \_)\}$;

PC-**critical,** if π contains an unprotected edge $T_i \xrightarrow{\mathsf{rw},\, \_} T_{i+1}$ and at least two adjacent unprotected edges with labels in $\{(\mathsf{ww}, \_), (\mathsf{rw}, \_)\}$;

PSI-**critical,** if:

1. π contains at least two unprotected rw edges; and
2. for every $T_i \xrightarrow{\mathsf{rw},\, x} T_{i+1}$, $T_j \xrightarrow{\mathsf{rw},\, y} T_{j+1} \in \pi$, if $i \neq j$, then $x \neq y$;

SI-**critical,** if:

1. π contains at least two adjacent unprotected rw edges; and
2. for every $T_i \xrightarrow{\mathsf{rw},\, x} T_{i+1}$, $T_j \xrightarrow{\mathsf{rw},\, y} T_{j+1} \in \pi$, if $i \neq j$, then $x \neq y$. ◀

The graphs of the executions in Figure 2 (with $\mathsf{level} = (\lambda T.\, \bot)$) contain critical cycles: (c) contains a PSI-critical cycle, and (d) contains an SI-critical cycle.

▶ **Theorem 5.** *For every wm and X, if $X \models wm$, then $\mathsf{DDG}(X)$ contains a cycle if and only if it contains a wm-critical cycle.*

From Theorem 5 and Lemma 3 we obtain our dynamic robustness criterion.

▶ **Corollary 6.** *For every wm and every X, if $X \models wm$ and $\mathsf{DDG}(X)$ contains no wm-critical cycle then $X.H \in \mathsf{hist}(\mathsf{SER})$.*

We note that the above robustness criterion for SI is a variant of an existing one [14, 15]. In our setting, it is just a consequence of our novel criterion for PSI.

To prove Theorem 5 we show how the axioms in Figure 3 impact the properties of edges and paths in dependency graphs. First, observe that there is a relation between wr, ww edges and the orders $<_{\mathsf{hb}}$ and $<_{\mathsf{ar}}$: for every X and every $T, S \in X.H$, the definitions ensure that

$$\begin{aligned} T \xrightarrow{\mathsf{wr},\_} S \in \mathsf{DDG}(X) &\implies T <_{\mathsf{hb}} S; \\ (T \xrightarrow{\mathsf{wr},\_} S \in \mathsf{DDG}(X) \vee T \xrightarrow{\mathsf{ww},\_} S \in \mathsf{DDG}(X)) &\implies T <_{\mathsf{ar}} S; \\ (X \models \textsc{Conflict} \wedge T \xrightarrow{\mathsf{ww},\_} S \in \mathsf{DDG}(X)) &\implies T <_{\mathsf{hb}} S. \end{aligned}$$

These implications let us show that, under certain conditions, if two transactions in a dependency graph are connected by a path, then they are also related by happens-before or arbitration.

▶ **Lemma 7.** *For any X, $s \in L^+$ and $T \xrightarrow{s}{}^+ S \in \mathsf{DDG}(X)$, if $X \models \textsc{SerTotal}$ then:*

1. *if all the rw and ww edges in $T \xrightarrow{s}{}^+ S$ are protected then $T <_{\mathsf{hb}} S$;*
2. *if all the rw edges in $T \xrightarrow{s}{}^+ S$ are protected then $T <_{\mathsf{ar}} S$;*
3. *if $X \models \textsc{Conflict}$ and all the rw edges in $T \xrightarrow{s}{}^+ S$ are protected then $T <_{\mathsf{hb}} S$.*

The following lemma shows that, if $T \xrightarrow{\mathsf{rw},\, x} S$, then T cannot happen-before S: in this case T would have to read a value at least as up-to-date as that written by S, contradicting the definition of anti-dependencies.

▶ **Lemma 8.** $\forall X. \forall x \in \mathsf{Obj}. \forall T, S \in X.H. T \xrightarrow{\mathsf{rw},\, x} S \in \mathsf{DDG}(X) \implies$
$S \not\leq_{\mathsf{hb}} T \wedge T \vdash \mathtt{read}(x, \_) \wedge S \vdash \mathtt{write}(x, \_)$.

Proof of Theorem 5. The *if* implication is obvious, so let us prove the *only if* implication. Suppose that the graph $\mathsf{DDG}(X)$ contains a cycle π'. From π' we can easily build a cycle

$$\pi = T_0 \xrightarrow{\lambda_0} T_1 \xrightarrow{\lambda_1} \dots \xrightarrow{\lambda_{n-1}} T_n \quad (\text{where } T_0 = T_n,\ n \geq 2) \tag{1}$$

in $\mathsf{DDG}(X)$ that is simple, chord-free and rw-minimal. The argument now is a case analysis on the *wm*. Here we consider only CC and PSI and defer the full proof to §A.

Case of $wm = \mathsf{CC}$**.** Lemma 7(2) implies that π contains at least one unprotected rw edge, for otherwise $T_0 <_{\mathsf{ar}} T_0$, contradicting the irreflexivity of $X.{<_{\mathsf{ar}}}$. Let this edge be $T_i \xrightarrow{\mathsf{rw},\_} T_{i+1}$. Then Lemma 8 ensures that $T_{i+1} \not<_{\mathsf{hb}} T_i$. Since π contains the non-empty path $T_{i+1} \xrightarrow{}{}^+ T_i$, Lemma 7(1) implies that on this path there is at least one unprotected edge $T_j \xrightarrow{\lambda} T_{j+1}$ with $\lambda \in \{(\mathsf{ww}, \_), (\mathsf{rw}, \_)\}$ and $i \neq j$. It follows that π is CC-critical.

Case of $wm = \mathsf{PSI}$**.** First we prove that the cycle π contains at least two unprotected edges rw edges. Since $X \models \mathsf{PSI}$, we know that $X \models \mathsf{CC}$. Thus, the previous argument ensures that the cycle π contains at least one unprotected rw edge, say $T_i \xrightarrow{\mathsf{rw},\, x} T_{i+1}$. Suppose that π contains exactly one such edge. Since $X \models \textsc{Conflict}$, Lemma 7(3) now ensures $T_{i+1} <_{\mathsf{hb}} T_i$. But by Lemma 8, $T_i \xrightarrow{\mathsf{rw},\, x} T_{i+1}$ implies $T_{i+1} \not<_{\mathsf{hb}} T_i$. The resulting contradiction shows that π must contain at least two unprotected rw edges.

Now we have to prove

$$\forall T_i \xrightarrow{\mathsf{rw},\, x} T_{i+1}, T_j \xrightarrow{\mathsf{rw},\, y} T_{j+1} \in \pi. i \neq j \implies x \neq y. \tag{2}$$

Suppose that π does not satisfy (2). Then, as $<_{\mathsf{ar}}$ is total, Definition 2 guarantees that we have either $T_{i+1} \xrightarrow{\mathsf{ww},x} T_{j+1}$ or $T_{j+1} \xrightarrow{\mathsf{ww},x} T_{i+1}$. Since π is a simple cycle, in the first case we contradict either that π is chord-free or that π is rw-minimal. In the second case, we have either (a) $T_{j+1} = T_i$ or (b) $T_{j+1} \neq T_i$. If (a) holds, then we contradict that π is rw-minimal, because $T_i \xrightarrow{\mathsf{rw},\_} T_{i+1} \in \pi$ and $T_i \xrightarrow{\mathsf{ww},\_} T_{i+1}$. If (b) holds, then the sub-path $T_{j+1} \xrightarrow{s}^{+} T_{i+1}$ of π contains at least two edges and it is chord-free by construction. But this contradicts $T_{j+1} \xrightarrow{\mathsf{ww},x} T_{i+1}$. It follows that π satisfies (2) above, and thus it is a PSI-critical cycle. ◀

4 Static Robustness Criteria

We now illustrate how the dynamic robustness criteria (Corollary 6) can be lifted to static criteria, which allow programmers to analyse the behaviour of their applications and which can serve as a basis for static analysis tools.

We define an ***application*** $\mathcal{A}$ by a set of ***transactional programs*** $\mathtt{f}_i$, giving the code of its transactions: $\mathcal{A} = \{\mathtt{f}_1, \ldots, \mathtt{f}_n\}$ (e.g., see Figure 1). As is standard in the database literature [15], this abstracts from the rest of the application logic to focus on the parts that directly interact with the database. We call a pair $I = (\mathtt{f}, \overline{\mathtt{v}})$ of a program and a vector of its actual parameters a ***program instance***. An ***application instance*** $\mathcal{I}$ is a set of program instances, and an ***annotated application instance*** is a pair $(\mathcal{I}, \mathsf{level}_S)$, where $\mathsf{level}_S : \mathcal{I} \to \{\mathsf{SER}, \perp\}$ defines which programs the programmer requested to execute under serializability. We first formulate criteria for checking the robustness of a particular annotated application instance, resulting from running a set of transactional programs with given parameters. We then sketch how these criteria can be generalised to whole applications.

We aim to illustrate the ideas for lifting dynamic robustness criteria to static ones in the simplest form. To this end, we abstract from the syntax of the programming language and assume that we are only given approximate information about the set of objects read or written by each transactional program. Namely, we assume a function rwsets that maps every program instance I to a triple $\mathsf{rwsets}(I) = (R^\diamond, W^\diamond, W^\square)$. Informally, $R^\diamond$ and $W^\diamond$ are the sets of all the objects that *may* be read or written in some execution of I, and $W^\square$ is a set of the objects that *must* be written in any execution of I, with the proviso that $W^\square \subseteq W^\diamond$. For instance, for $I = (\mathtt{StoreBid}, \langle iId_1, 7\rangle)$ (Figure 1) we have

$$\mathsf{rwsets}(I) = (\{\mathrm{ITEMS}(iId_1).\mathrm{nbids}\}, \{\mathrm{ITEMS}(iId_1).\mathrm{nbids}, \mathrm{BIDS}(*).*\}, \{\mathrm{ITEMS}(iId_1).\mathrm{nbids}\})$$

where $*$ means "all fields" or "all rows".

To formalise the meaning of the read/write sets, we define a relation that determines if a history can be produced by a given $\mathcal{I}$. We let $T \Vdash I$ for $\mathsf{rwsets}(I) = (R^\diamond, W^\diamond, W^\square)$, if:

(i) $T \vdash \mathtt{write}(x, \_) \implies x \in W^\diamond$;
(ii) $T \vdash \mathtt{read}(x, \_) \implies x \in R^\diamond$;
(iii) $x \in W^\square \implies T \vdash \mathtt{write}(x, \_)$.

We lift the relation $\Vdash$ to annotated histories and annotated application instances:

$$(H, \mathsf{level}) \Vdash (\mathcal{I}, \mathsf{level}_S) \iff \forall T \in H. \exists I \in \mathcal{I}. T \Vdash I \wedge \mathsf{level}_S(I) = \mathsf{level}(T).$$

Note that the definition of $\Vdash$ allows multiple transactions in H to be associated to a single I in $\mathcal{I}$. For example, we have $(H, (\lambda T. \perp)) \Vdash (\mathcal{I}, (\lambda I. \perp))$ for the history H in Figure 5 and

$$\mathcal{I} = \{(\mathtt{RegUser}, Alice), (\mathtt{ViewItem}, iId_1), (\mathtt{StoreBid}, \langle iId_1, 7\rangle), (\mathtt{StoreBid}, \langle iId_2, 10\rangle)\}. \quad (3)$$

We formulate our robustness criteria by adapting modal transition systems [20].

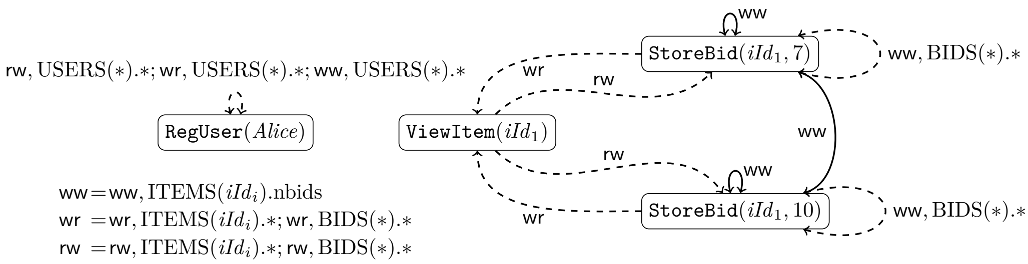


Figure 6 The static dependency graph $\mathsf{SDG}(\mathcal{I})$ of the application instance $\mathcal{I}$ defined in (3). We draw may edges with dashed arrows, and must edges with solid arrows.

▶ **Definition 9.** The ***static dependency graph*** of an application instance $\mathcal{I}$ is a triple $\mathsf{SDG}(\mathcal{I}) = (\mathcal{I}, \dashrightarrow, \longleftrightarrow)$, where the relations $\dashrightarrow$ and $\longleftrightarrow$ are defined as follows. For every $I, J \in \mathcal{I}$, if $\mathsf{rwsets}(I) = (W_I^{\diamond}, R_I^{\diamond}, W_I^{\square})$ and $\mathsf{rwsets}(J) = (W_J^{\diamond}, R_J^{\diamond}, W_J^{\square})$, then:

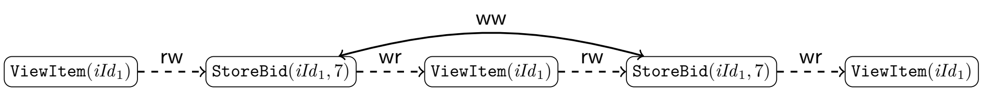
$$I \overset{\mathsf{wr},\, x}{\dashrightarrow} J \iff x \in W_I^{\diamond} \cap R_J^{\diamond}; \qquad I \overset{\mathsf{ww},\, x}{\dashrightarrow} J \iff x \in W_I^{\diamond} \cap W_J^{\diamond};$$
$$I \overset{\mathsf{rw},\, x}{\dashrightarrow} J \iff x \in R_I^{\diamond} \cap W_J^{\diamond}; \qquad I \xleftrightarrow{\mathsf{ww}, x} J \iff x \in W_I^{\square} \cap W_J^{\square}.$$

Figure 6 shows the static dependency graph of the $\mathcal{I}$ in (3). Informally, the edges of the static dependency graph $\mathsf{SDG}(\mathcal{I})$ describe possible dependencies between transactions in executions produced by $\mathcal{I}$: an edge $I \overset{\lambda}{\dashrightarrow} J$ represents a dependency that may exist, and an edge $I \overset{\lambda}{\longleftrightarrow} J$ a dependency that must exists. Formally, given an annotated application instance $(\mathcal{I}, \mathsf{level}_S)$, we say that the pair $(\mathsf{DDG}(X), X.\mathsf{level})$ is ***over-approximated*** by the pair $(\mathsf{SDG}(\mathcal{I}), \mathsf{level}_S)$, written $(\mathsf{DDG}(X), X.\mathsf{level}) \lhd (\mathsf{SDG}(\mathcal{I}), \mathsf{level}_S)$, if for some total function $f : X.H \to \mathcal{I}$ we have:

1. $\forall T \xrightarrow{\lambda} S \in \mathsf{DDG}(X).\, f(T) \overset{\lambda}{\dashrightarrow} f(S) \in \mathsf{SDG}(\mathcal{I})$;
2. $\forall I \overset{\lambda}{\longleftrightarrow} J \in \mathsf{SDG}(\mathcal{I})$.
   $\forall T \in f^{-1}(I). \forall S \in f^{-1}(J).\, T \xrightarrow{\lambda} S \in \mathsf{DDG}(X) \vee S \xrightarrow{\lambda} T \in \mathsf{DDG}(X)$; and
3. $\mathsf{level}(T) = \mathsf{level}_S(f(T))$.

▶ **Lemma 10.** $\forall X. \forall(\mathcal{I}, \mathsf{level}_S).\, (X.H, X.\mathsf{level}) \Vdash (\mathcal{I}, \mathsf{level}_S) \implies$
$(\mathsf{DDG}(X), \mathsf{level}) \lhd (\mathsf{SDG}(\mathcal{I}), \mathsf{level}_S)$.

We now formulate our static robustness criteria by using the same notions of paths and cycles for static dependency graphs as for dynamic ones (§3). Given a pair $(\mathcal{I}, \mathsf{level}_S)$ and a cycle in $\pi \in \mathsf{SDG}(\mathcal{I})$ among program instances $I_0, I_1, \ldots, I_n$ (where $I_0 = I_n$), we say that an rw edge $I_i \overset{\mathsf{rw},\, x}{\dashrightarrow} I_{i+1} \in \pi$ is ***critical in*** π, if for all I_l, I_m in π such that $l \neq m$ and for all $t, t' \in D^\star$ such that $I_l \overset{t}{\dashrightarrow}{}^\star I_i \in \pi$ and that $I_{i+1} \overset{t'}{\dashrightarrow}{}^\star I_m \in \pi$, we have $\neg(I_l \xleftrightarrow{\mathsf{ww}, -} I_m)$. For example, the graph in Figure 6 contains the following cycle π, in which the left-most rw edge is critical, while the right-most rw edge is not critical:



(4)

▶ **Definition 11.** Given a pair $(\mathcal{I}, \mathsf{level}_S)$, an edge $I_i \overset{\lambda}{\dashrightarrow} I_{i+1} \in \mathsf{SDG}(\mathcal{I})$ is ***unprotected*** if either $\mathsf{level}_S(I_i) \neq \mathsf{SER}$ or $\mathsf{level}_S(I_{i+1}) \neq \mathsf{SER}$. A cycle $\pi \in \mathsf{SDG}(\mathcal{I})$ among program instances $I_0, I_1, \ldots, I_n$ (where $I_0 = I_n$) is:

CC-**critical,** if π contains an unprotected edge $I_i \xrightarrow{\mathsf{rw}, \_} I_{i+1}$ and an unprotected edge $I_j \xrightarrow{\lambda} I_{j+1}$ with $i \neq j$ and $\lambda \in \{(\mathsf{ww}, \_), (\mathsf{rw}, \_)\}$;

PC-**critical,** if π contains an unprotected edge $I_i \xrightarrow{\mathsf{rw}, \_} I_{i+1}$ and at least two adjacent unprotected edges with labels in $\{(\mathsf{ww}, \_), (\mathsf{rw}, \_)\}$. ;

PSI-**critical,** if:

1. π contains at least two unprotected critical rw edges; and
2. for every $I_i \xrightarrow{\mathsf{rw}, x} I_{i+1}$, $I_j \xrightarrow{\mathsf{rw}, y} I_{j+1} \in \pi$, if $i \neq j$, then $x \neq y$;

SI-**critical,** if:

1. π contains at least two adjacent unprotected critical rw edges; and
2. for every $I_i \xrightarrow{\mathsf{rw}, x} I_{i+1}$, $I_j \xrightarrow{\mathsf{rw}, y} I_{j+1} \in \pi$, if $i \neq j$, then $x \neq y$. ◀

Note that, unlike a critical cycle in a dynamic dependency graph (Definition 4), a critical cycle in a static graph does not have to be simple. The following lemma states that $\lhd$ preserves critical cycles.

▶ **Lemma 12.** *For every wm, (G, level) and (F, level_S) such that $(G, \mathsf{level}) \lhd (F, \mathsf{level}_S)$, if G contains a wm-critical cycle, then F contains a wm-critical cycle.*

Lemmas 10 and 12 (which are proven in §A) and Corollary 6 establish our static criteria.

▶ **Theorem 13.** *For every (H, level), $(\mathcal{I}, \mathsf{level}_S)$ and wm, if $\mathsf{SDG}(\mathcal{I})$ contains no wm-critical cycles and $(H, \mathsf{level}) \in \mathsf{hist}(wm)$, then whenever $(H, \mathsf{level}) \Vdash (\mathcal{I}, \mathsf{level}_S)$, we have $H \in \mathsf{hist}(\mathsf{SER})$.*

For example, let $\mathsf{level}_S = (\lambda I. \bot)$ and consider $\mathcal{I}$ defined by (3). The corresponding static dependency graph in Figure 6 contains PSI-critical cycles, one of which is obtained by following twice the loop $\mathtt{RegUser}(Alice) \xrightarrow{\mathsf{rw}, \mathrm{USERS}(\_).\mathrm{name}} \mathtt{RegUser}(Alice)$. Indeed, as we explained in §2, the annotated instance $(\mathcal{I}, \mathsf{level}_S)$ is not robust against PSI, because it can produce the write skew anomaly in Figure 2(d). Now let $\mathsf{level}'_S(\mathtt{RegUser}, Alice) = \mathsf{SER}$ and $\mathsf{level}'_S(\_) = \bot$ otherwise. Figure 6 contains the static dependency graph corresponding to the annotated instance $(\mathcal{I}, \mathsf{level}'_S)$. This graph does not contain PSI-critical cycles. To see why, observe that in the graph there are only two kinds of cycles: the ones due to the self-loop on the node $\mathtt{RegUser}(Alice)$, and the ones that connect nodes in $\{\mathtt{StoreBid}(iId_1, 7), \mathtt{StoreBid}(iId_1, 10), \mathtt{ViewItem}(iId_1)\}$. The cycles of the first kind contain only protected rw edges thanks to level'_S, while the cycles of the second kind contain at most one critical rw edge, as sketched in (4) above. It follows that no cycle is PSI-critical, and thus by executing only the `RegUser` transaction on serializability, we make $\mathcal{I}$ robust. However, the graph contains a CC-critical cycle, namely the one shown in (4) above. It is CC-critical for its two rw edges are unprotected. As we explained in §2, under CC $\mathcal{I}$ may produce the lost update anomaly in Figure 2(b), and it is unsafe to run $\mathcal{I}$ over a CC database.

Analysing Whole Applications. The static criteria in Theorem 13 allow a programmer to analyse the robustness of a given application instance. Analysing an application completely using the theorem requires considering an infinite number of its instances, a task best done by an automatic static analysis tool. We now sketch how ideas from abstract interpretation [12, 24] can be used to finitely represent and analyse the set of all instances of an application. In the future, this can pave the way to automating our robustness criteria in static analysis tools. Due to space constraints, we only present the concepts by an example.

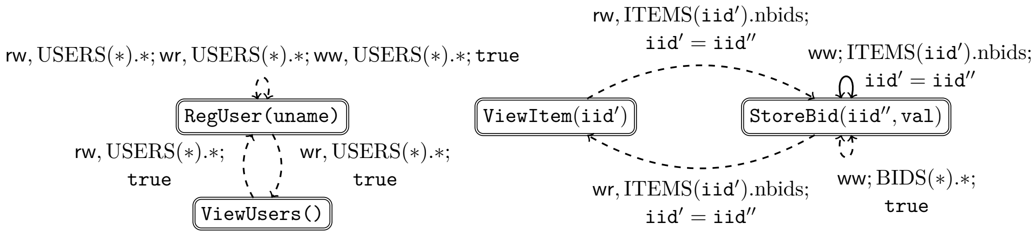
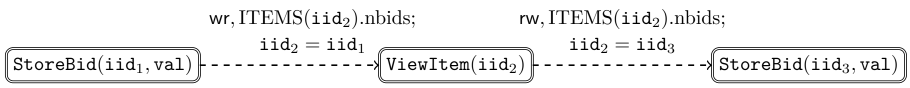


Figure 7 The summary dependency graph $\mathsf{SDG}(\mathcal{A})$, where $\mathcal{A}$ contains all the transactional programs in Figure 1.

We associate an application $\mathcal{A}$ with a ***summary dependency graph*** $\mathsf{SDG}(\mathcal{A})$ that summarises the static graphs of all the instances of $\mathcal{A}$. In Figure 7 we show the summary dependency graph $\mathsf{SDG}(\mathcal{A})$ for the application in Figure 1. Every program in $\mathcal{A}$ yields a ***summary node*** in the graph $\mathsf{SDG}(\mathcal{A})$, representing all instances of the program. Every edge in $\mathsf{SDG}(\mathcal{A})$ is a ***summary edge***, summarising the possible dependencies between the corresponding programs. It is annotated by a constraint relating the actual parameters of the incident programs between which the dependency exists. For example, the summary edge from `ViewItem` to `StoreBid` in Figure 7 means that for every instance $\mathcal{I}$ of $\mathcal{A}$ we have $\texttt{StoreBid}(iId_1, \_) \xrightarrow{\mathsf{rw}, x} \texttt{ViewItem}(iId_2)$ in $\mathsf{SDG}(\mathcal{I})$ iff $x \in \{\text{BIDS}(*).\text{iId}, \text{BIDS}(*).\text{val}, \text{ITEMS}(iId_1).\text{nbids}\}$ and $iId_1 = iId_2$. Similarly, the ww edge incident to `StoreBid` means that we have $\texttt{StoreBid}(iId_1, \_) \xleftrightarrow{\mathsf{ww}, x} \texttt{StoreBid}(iId_2, \_)$ in $\mathsf{SDG}(\mathcal{I})$ iff $x = \text{ITEMS}(iId_1).\text{nbids}$ and $iId_1 = iId_2$.

Definition 11 carries over to summary graphs of applications by taking into account the constraints on summary edges when checking whether a given rw edge is critical in a given cycle π, and whether the objects that appear on the rw edges of π are different. For example, consider the following cycle in the graph in Figure 7:



We consider the rw edge on this cycle not critical. This is because the constraints on the edges in the cycle imply $\texttt{iid}_1 = \texttt{iid}_3$, which satisfies the constraint on the must ww edge between `StoreBid` programs in Figure 7. For any annotated instance $(\mathcal{I}, \mathsf{level}_S)$ of the application $\mathcal{A}$ in Figure 1, using the adjusted Definition 11 we can check that, if level_S maps the instances of `RegUser` and `ViewUser` in $\mathcal{I}$ to SER, then $(\mathcal{I}, \mathsf{level}_S)$ is robust against PSI[1].

5 Related Work

In the setting of databases, application robustness was first investigated by Fekete et al. [15], who proposed a criterion for robustness against snapshot isolation (SI) [6]. Fekete then extended the criterion to SI databases allowing the programmer to request serializability for certain transactions [14], a mechanism that we also consider. Our criterion is formulated in a way similar to that of Fekete et al., using dependency graphs [2]. However, in contrast to

[1] Marking `ViewUser` as SER is actually unnecessary to make this application robust under PSI, because the graph $\mathsf{SDG}(\mathcal{A})$ contains an edge $\texttt{ViewUser}() \xrightarrow{\mathsf{rw}, \text{USERS}(*).*} \texttt{RegUser}(\texttt{uname})$ which does not exist in the dependency graph of any execution of $\mathcal{A}$. This can be addressed by a more precise static analysis.

their work, we consider more subtle models of parallel snapshot isolation, prefix consistency and causal consistency, which allow more anomalies than SI. The method we use is also different from that of Fekete et al. They consider an operational specification of SI [6], which makes the proof of the robustness criterion highly involved. In contrast, we benefit from using declarative specifications that achieve conciseness by exploiting atomic visibility of transactions [11]. This allows us to come up with robustness criteria more systematically.

Robustness has also been investigated for applications running on weak shared-memory models of common multiprocessors and programming languages (e.g., [8]). However, this line of work has not considered applications using transactions. Transactions complicate the consistency model semantics, which makes establishing robustness criteria more challenging.

Serializability of transactions in an application simplifies establishing its correctness properties, but is not necessary for this. Thus, an alternative approach to establishing application correctness is to prove its desired properties directly, without requiring the transactions to produce only serializable behaviours. Corresponding methods have been proposed for ANSI SQL isolation levels and SI by Lu et al. [22], and for PSI and some of other recent models by Gotsman et al. [17]. Such methods are complementary to ours: the conditions they require can be satisfied by more applications, but are more difficult to check than robustness.

6 Conclusion

In this paper we have made the first steps towards understanding the impact of recently-proposed transactional consistency models for large-scale databases on the correctness properties of applications using them. To this end, we have proposed criteria for checking when an application using a weak consistency model exhibits only strongly consistent behaviours. This enables programmers to check that application correctness will be preserved for a particular choice of a consistency model or transactions to be executed under serializability.

The robustness result of Fekete et al. for SI has previously given rise to automatic tools for statically detecting anomalies in applications [18]. Our work could form a basis for similar advances in databases providing weaker consistency models. Our dynamic robustness criteria are also of an independent interest: apart from serving as a basis for static analysis, such criteria can be used to optimise run-time monitoring algorithms [23, 27].

In establishing our robustness criteria, we have followed a systematic approach that exploits axiomatic specifications [11]: using the axioms of a consistency model, we have characterised the cycles allowed in dependency graphs of executions on the model, and exploited the characterisations to provide sound static analysis techniques. We hope that this method will be applicable to other consistency models being proposed for large-scale databases.

References

1 D. Abadi. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *IEEE Computer*, 45(2), 2012.

2 A. Adya. *Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions*. Ph.D., MIT, Cambridge, MA, USA, Mar. 1999.

3 M. S. Ardekani, P. Sutra, and M. Shapiro. Non-monotonic snapshot isolation: Scalable and strong consistency for geo-replicated transactional systems. In *SRDS*, 2013.

4 P. Bailis, A. Davidson, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Highly Available Transactions: virtues and limitations. In *VLDB*, 2014.

5 P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Scalable atomic visibility with RAMP transactions. In *SIGMOD*, 2014.

6 H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ANSI SQL isolation levels. In *SIGMOD*, 1995.

7 P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. 1987.

8 A. Bouajjani, E. Derevenetc, and R. Meyer. Checking and enforcing robustness against TSO. In *ESOP*, 2013.

9 S. Burckhardt, D. Leijen, M. Fähndrich, and M. Sagiv. Eventually consistent transactions. In *ESOP*, 2012.

10 S. Burckhardt, S. Leijen, J. Protzenko, and M. Fähndrich. Global sequence protocol: A robust abstraction for replicated shared state. In *ECOOP*, 2015.

11 A. Cerone, G. Bernardi, and A. Gotsman. A framework for transactional consistency models with atomic visibility. In *CONCUR*, 2015.

12 P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, 1977.

13 G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *SOSP*, 2007.

14 A. Fekete. Allocating isolation levels to transactions. In *PODS*, 2005.

15 A. Fekete, D. Liarokapis, D. J. O'Neil, P. O'Neil, and D. Shasha. Making snapshot isolation serializable. *ACM Trans. Database Syst.*, 30(2), 2005.

16 S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2), 2002.

17 A. Gotsman, H. Yang, C. Ferreira, M. Najafzadeh, and M. Shapiro. 'Cause I'm strong enough: reasoning about consistency choices in distributed systems. In *POPL*, 2016.

18 S. Jorwekar, A. Fekete, K. Ramamritham, and S. Sudarshan. Automating the detection of snapshot isolation anomalies. In *VLDB*, 2007.

19 A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2), 2010.

20 K. G. Larsen and B. Thomsen. A modal process logic. In *LICS*, 1988.

21 W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In *SOSP*, 2011.

22 S. Lu, A. J. Bernstein, and P. M. Lewis. Correct execution of transactions at different isolation levels. *IEEE Trans. Knowl. Data Eng.*, 16(9), 2004.

23 D. R. K. Ports and K. Grittner. Serializable snapshot isolation in PostgreSQL. *Proc. VLDB Endow.*, 5(12), Aug. 2012.

24 M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Trans. Program. Lang. Syst.*, 24(3):217–298, 2002.

25 Y. Sovran, R. Power, M. K. Aguilera, and J. Li. Transactional storage for geo-replicated systems. In *SOSP*, 2011.

26 D. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, and M. K. Aguilera. Transactions with consistency choices on geo-replicated cloud storage. Technical Report MSR-TR-2013-82, Microsoft Research, 2013.

27 K. Zellag and B. Kemme. Consistency anomalies in multi-tier architectures: Automatic detection and prevention. *The VLDB Journal*, 23(1), 2014.

A Additional Proofs

▶ **Lemma 7.** *For every X, every $s \in L^+$ and every $T \xrightarrow{s}{}^+ S \in \mathsf{DDG}(X)$ if $X \models$ SerTotal then we have:*

1. *if all the* rw *and* ww *edges in $T \xrightarrow{s}{}^+ S$ are protected then $T <_{\mathsf{hb}} S$;*
2. *if all the* rw *edges in $T \xrightarrow{s}{}^+ S$ are protected then $T <_{\mathsf{ar}} S$;*
3. *if $X \models$ Conflict and all the* rw *edges in $T \xrightarrow{s}{}^+ S$ are protected then $T <_{\mathsf{hb}} S$.*

Proof. In all the cases the argument is an induction on the length of the string s, the base case being $|s| = 1$. In the first and third case of the lemma, the inductive reasoning relies on the following implication,

$$T \xrightarrow{\lambda} S \in \mathsf{DDG}(X) \wedge T \xrightarrow{\lambda} S \text{ protected} \implies T <_{\mathsf{hb}} S$$

which holds thanks to the hypothesis that $X \models$ SerTotal. ◀

▶ **Lemma 8.** $\forall X. \forall x \in \mathsf{Obj}. \forall T, S \in X.H. T \xrightarrow{\mathsf{rw},\, x} S \in \mathsf{DDG}(X) \implies$
$S \not\leq_{\mathsf{hb}} T \wedge T \vdash \mathtt{read}(x, \_) \wedge S \vdash \mathtt{write}(x, \_)$.

Proof. Fix an execution $X = ((H, \_), <_{\mathsf{hb}}, <_{\mathsf{ar}})$, and pick $T, S \in H$ such that $T \xrightarrow{\mathsf{rw},\, x} S \in \mathsf{DDG}(X)$ for some $x \in \mathsf{Obj}$.

Definition 2 and $T \xrightarrow{\mathsf{rw},\, x} S$ imply that $T \neq S$, so now to prove the required $S \not\leq_{\mathsf{hb}} T$ it is enough to show just $S \not<_{\mathsf{hb}} T$.

Definition 2(2) ensures that one of the following two cases holds:

1. $T \vdash \mathtt{read}(x, \_)$, $S \vdash \mathtt{write}(x, \_)$ and $\mathsf{before}(S, <_{\mathsf{hb}}, \mathtt{write}\ x) = \emptyset$,
2. $T' \xrightarrow{\mathsf{wr},\, x} T$ and $T' \xrightarrow{\mathsf{ww},\, x} S$ for some $T' \in H$.

In the first case we have $T \vdash \mathtt{read}(x, \_)$, $S \vdash \mathtt{write}(x, \_)$ and $\mathsf{before}(T, <_{\mathsf{hb}}, \mathtt{write}\ x) = \emptyset$. The last equality implies $S \not<_{\mathsf{hb}} T$.

In the second case we have $T' \xrightarrow{\mathsf{wr},\, x} T$ and $T' \xrightarrow{\mathsf{ww},\, x} S$ for some $T' \in H$, thus according to Definition 2 we have that

$$T \vdash \mathtt{read}(x, \_), \quad S \vdash \mathtt{write}(x, \_) \quad \text{and} \quad T' \vdash \mathtt{write}(x, \_),$$

moreover $T' = \max(B, <_{\mathsf{ar}})$ where $B = \mathsf{before}(T, <_{\mathsf{hb}}, \mathtt{write}\ x)$ and $T' <_{\mathsf{ar}} S$. We now prove that $S \not<_{\mathsf{hb}} T$. Consider the set B, since $S \vdash \mathtt{write}(x, \_)$ we have $S \not<_{\mathsf{hb}} T$ if and only if $S \notin B$. But $S \notin B$, for otherwise $T' = \max(B, <_{\mathsf{ar}})$ implies $S <_{\mathsf{ar}} T'$, and so from $T' <_{\mathsf{ar}} S$ we obtain $T' <_{\mathsf{ar}} S <_{\mathsf{ar}} T'$, which contradicts the irreflexivity of $<_{\mathsf{ar}}$. ◀

▶ **Lemma 14.** *For every X such that $X \models$ SerTotal$\wedge$Prefix, a cycle in $\mathsf{DDG}(X)$ contains at least two unprotected edges if and only if it contains at least two adjacent unprotected edges with labels in $\{(\mathsf{ww}, \_), (\mathsf{rw}, \_)\}$.*

Proof. The *only if* implication is obvious, so let us prove the *if* implication. Fix a cycle $\pi \in \mathsf{DDG}(X)$ that contains at least two unprotected edges. One such cycle exists by assumption. We assume without loss of generality that π is of the form

$$\begin{aligned} \pi = S_0 &\xrightarrow{s_0}{}^\star T_0 \xrightarrow{\lambda_0} S_1 \\ &\xrightarrow{s_1}{}^\star T_1 \xrightarrow{\lambda_1} S_2 \\ &\quad \dots \\ &\xrightarrow{s_{n-1}}{}^\star T_{n-1} \xrightarrow{\lambda_{n-1}} S_n = S_0, \end{aligned} \tag{5}$$

where $n \geq 2$, all the paths with strings s_i's contain only edges whose labels are in $\{(\mathsf{wr}, \_)\}$ or are protected, while the edges $T_i \xrightarrow{\lambda_i} S_{i+1}$ are unprotected and $\lambda_i \in \{(\mathsf{ww}, \_), (\mathsf{rw}, \_)\}$. We have to show $S_i = T_i$ for some i. Suppose that this is not true, so that $S_i \neq T_i$ for every $i = 0..(n-1)$.

We use this assumption to show that for every $i \in [0; n-1]$ we have $S_i <_{\mathsf{ar}} S_{i+1}$. This is enough to prove the lemma, because it implies that $S_0 <_{\mathsf{ar}} S_1 <_{\mathsf{ar}} \ldots <_{\mathsf{ar}} S_{n-1} <_{\mathsf{ar}} S_0$, which contradicts the irreflexivity of $<_{\mathsf{ar}}$.

Pick an $i \in [0; n-1]$. Our assumption that $S_i \neq T_i$ implies that $s_i \in L^+$, and thus $S_i \xrightarrow{s_i}{}^{+} T_i$. The properties of the edges in s_i and Lemma 7(1), imply that $S_i <_{\mathsf{hb}} T_i$. Now the argument depends on λ_i. If $\lambda_i = (\mathsf{rw}, x_i)$, then Lemma 8 ensures that

$$S_{i+1} \not\leq_{\mathsf{hb}} T_i \tag{6}$$

Now $S_i <_{\mathsf{hb}} T_i$ implies that $S_i \neq S_{i+1}$, and so the totality of $<_{\mathsf{ar}}$ ensures that

$$S_i <_{\mathsf{ar}} S_{i+1} \quad \text{or} \quad S_{i+1} <_{\mathsf{ar}} S_i.$$

If $S_{i+1} <_{\mathsf{ar}} S_i$ then the hypothesis $X \models \textsc{Prefix}$ and $S_i <_{\mathsf{hb}} T_i$ imply that $S_{i+1} <_{\mathsf{hb}} T_i$, which contradicts (6) above. Hence, the required $S_i <_{\mathsf{ar}} S_{i+1}$.

If $\lambda_i = (\mathsf{ww}, x_i)$, then $T_i <_{\mathsf{ar}} S_{i+1}$. Since $S_i <_{\mathsf{hb}} T_i$ the set inclusion $<_{\mathsf{hb}} \subseteq <_{\mathsf{ar}}$ ensures that $S_i <_{\mathsf{ar}} T_i$, and thus the required $S_i <_{\mathsf{ar}} S_{i+1}$. ◀

▶ **Lemma 15.** *For every X such that $X \models \textsc{SerTotal} \wedge \textsc{Conflict} \wedge \textsc{Prefix}$, a cycle in $\mathsf{DDG}(X)$ contains at least two unprotected* rw *edges if and only if it contains at least two adjacent unprotected* rw *edges.*

Proof. The proof is virtually the same of Lemma 14, but it exploits the property that

$$X \models \textsc{Conflict} \implies (T \xrightarrow{\mathsf{ww},\_} S \in \mathsf{DDG}(X) \implies T <_{\mathsf{hb}} S)$$

in order to ensure that all the λ's are labels $(\mathsf{rw}, \_)$, and relies on Lemma 7(3) instead of Lemma 7(1). ◀

▶ **Theorem 5**. *For every wm and X, if $X \models wm$, then $\mathsf{DDG}(X)$ contains a cycle if and only if it contains a wm-critical cycle.*

Proof. The *if* implication is obvious, so let us prove the *only if* implication. Suppose that the graph $\mathsf{DDG}(X)$ contains a cycle π'. From π' we can easily build a cycle

$$\pi = T_0 \xrightarrow{\lambda_0} T_1 \xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_{n-1}} T_n \quad (\text{where } T_0 = T_n) \tag{7}$$

in $\mathsf{DDG}(X)$ that is simple, chord-free and rw-minimal. The argument now is a case analysis on *wm*.

Suppose $wm = \mathsf{CC}$. The irreflexivity of $X.<_{\mathsf{ar}}$ ensures that $T_0 \not<_{\mathsf{ar}} T_0$. Hence, since $T_0 \xrightarrow{\lambda_0 \ldots \lambda_0 \ldots \lambda_n}{}^{+} T_1$ Lemma 7(2) implies that π contains at least one rw edge that is unprotected, for otherwise $T_0 <_{\mathsf{ar}} T_0$. Let this edge be $T_i \xrightarrow{\mathsf{rw},\_} T_{i+1}$. Then Lemma 8 ensures that $T_{i+1} \not<_{\mathsf{hb}} T_i$. Since π contains the non-empty path $T_{i+1} \longrightarrow^{+} T_i$ Lemma 7(1) implies that in $T_{i+1} \longrightarrow^{+} T_i$ there is at least one unprotected edge $T_j \xrightarrow{\lambda} T_{j+1}$ with $\lambda \in \{(\mathsf{ww}, \_), (\mathsf{rw}, \_)\}$ and $i \neq j$. It follows that π is CC-critical.

Suppose that $wm = \mathsf{PC}$. The previous argument ensures that π contains at least two unprotected edges, one of which is a rw edge. An application of Lemma 14 lets us prove that that π contains at least two adjacent unprotected edges, thus it is PC-critical.

Suppose that $wm = \mathsf{PSI}$. First we prove that the cycle π contains at least two unprotected edges rw edges. Since $X \models \mathsf{PSI}$, we know that $X \models \mathsf{CC}$. Thus, the argument for CC ensures that the cycle π contains at least one unprotected rw edge, say $T_i \xrightarrow{\mathsf{rw},\, x} T_{i+1}$. Suppose that π contains exactly one such edge. Since $X \models$ Conflict, Lemma 7(3) now implies that $T_0 <_{\mathsf{hb}} T_i$ and $T_{i+1} <_{\mathsf{hb}} T_n = T_0$. By the transitivity of $<_{\mathsf{hb}}$ we have $T_{i+1} <_{\mathsf{hb}} T_i$. But this contradicts Lemma 8, which establishes $T_{i+1} \not<_{\mathsf{hb}} T_i$. Thus, π must contain at least two unprotected rw edges. Now we have to prove

$$\forall T_i \xrightarrow{\mathsf{rw},\, x} T_{i+1},\, T_j \xrightarrow{\mathsf{rw},\, y} T_{j+1} \in \pi.\, i \neq j \implies x \neq y. \tag{8}$$

Suppose now that π does not satisfy (2). Then as $<_{\mathsf{ar}}$ is total, Definition 2(2) guarantees that we have either $T_{i+1} \xrightarrow{\mathsf{ww},x} T_{j+1}$ or $T_{j+1} \xrightarrow{\mathsf{ww},x} T_{i+1}$. Since π is a simple cycle, in the first case we contradict either that π is chord-free, or that π is rw-minimal. In the second case note that either (a) $T_{j+1} = T_i$ or (b) $T_{j+1} \neq T_i$ If (a) holds then we contradict that π is rw-minimal, because $T_i \xrightarrow{\mathsf{rw},\_} T_{i+1} \in \pi$ and $T_i \xrightarrow{\mathsf{ww},\_} T_{i+1}$. If (b) holds, then the sub-path $T_{j+1} \xrightarrow{s}^{+} T_{i+1}$ of π contains at least 2 edges and it is chord-free by construction. But this contradicts $T_{j+1} \xrightarrow{\mathsf{ww},x} T_{i+1}$. It follows that π satisfies (2) above, and thus it is a PSI-critical cycle.

Suppose that $wm = \mathsf{SI}$. In this case $X \models \mathsf{PSI}$, thus the argument for $wm = \mathsf{PSI}$ ensures that the graph $\mathsf{DDG}(X)$ contains a cycle π' such that π' is chord-free, rw-minimal, it contains at least two unprotected rw edges. Lemma 15 implies that π' contains at least two adjacent unprotected rw edges, thus π' is SI-critical. ◀

Recall the definition of the relation $\Vdash$, we let $T \Vdash I$ whenever $\mathsf{rwsets}(I) = (R^{\diamond}, W^{\diamond}, W^{\square})$ and we have:

1. $T \vdash \mathtt{write}(x, \_)$ implies $x \in W^{\diamond}$;
2. $T \vdash \mathtt{read}(x, \_)$ implies $x \in R^{\diamond}$; and
3. $x \in W^{\square}$ implies $T \vdash \mathtt{write}(x, \_)$.

and $(H, \mathsf{level}) \Vdash (\mathcal{I}, \mathsf{level}_{\mathrm{S}})$ whenever

$$\forall T \in H.\, \exists I \in \mathcal{I}.\, T \Vdash I \wedge \mathsf{level}_{\mathrm{S}}(I) = \mathsf{level}(T)$$

Now we prove that the relation $\Vdash$ is sufficient to show that the dynamic dependency graph of an execution is over-approximated by a static dependency graph.

▶ **Lemma 10.** $\forall X.\, \forall(\mathcal{I}, \mathsf{level}_{\mathrm{S}}).\, (X.H, X.\mathsf{level}) \Vdash (\mathcal{I}, \mathsf{level}_{\mathrm{S}}) \implies$
$(\mathsf{DDG}(X), \mathsf{level}) \lhd (\mathsf{SDG}(\mathcal{I}), \mathsf{level}_{\mathrm{S}})$.

Proof. Fix an execution X and a pair $(\mathcal{I}, slevel)$ such that $(X.H, X.\mathsf{level}) \Vdash (\mathcal{I}, \mathsf{level}_{\mathrm{S}})$. The definition of $\lhd$ requires us to exhibit an $f : X.H \to \mathcal{I}$ such that the following conditions are true

(i) $T \overset{\lambda}{\dashrightarrow} S \in \mathsf{DDG}(X) \implies f(T) \overset{\lambda}{\dashrightarrow} f(S) \in \mathsf{SDG}(\mathcal{I})$;
(ii) $I \overset{\lambda}{\longleftrightarrow} J \in \mathsf{SDG}(\mathcal{I}) \implies$
$\forall T \in f^{-1}(I).\, \forall S \in f^{-1}(J).\, T \xrightarrow{\lambda} S \in \mathsf{DDG}(X) \vee S \xrightarrow{\lambda} T \in \mathsf{DDG}(X)$;
(iii) $\mathsf{level}(T) = \mathsf{level}_{\mathrm{S}}(f(T))$.

Let

$$\hat{f} = \{(T, I) \mid T \in X.H \wedge I \in \mathcal{I} \wedge T \Vdash I \wedge \mathsf{level}(T) = \mathsf{level}_{\mathrm{S}}(I)\}$$

If $\hat{f}$ is not a function, then it is routine work to construct an f that is a function, by picking for every T just one pair $(T, I) \in f$. Otherwise let $f = \hat{f}$. We have by construction that f is a function, moreover it is total, because the hypothesis $(X.H, X.\mathsf{level}) \Vdash (\mathcal{I}, \mathsf{level}_{\mathrm{S}})$ ensures that for every $T \in X.H$ there exists an $I \in \mathcal{I}$ such that $T \Vdash I \wedge \mathsf{level}(T) = \mathsf{level}_{\mathrm{S}}(I)$. Note also that f satisfies part (iii) by construction.

To prove part (i), suppose that $T \overset{\lambda}{\dashrightarrow} S \in \mathsf{DDG}(X)$ for some $S, T \in X.H$. We reason by case analysis. If $\lambda = (\mathsf{wr}, x)$, then Definition 2 implies that $T \vdash \mathtt{write}(x, \_)$ and $S \vdash \mathtt{read}(x, \_)$. The definition of $\Vdash$ now ensures that $x \in R^{\diamond}$ and $x \in W^{\diamond}$, where $\mathsf{rwsets}(f(T)) = (R^{\diamond}, \_, \_)$ and $\mathsf{rwsets}(f(S)) = (\_, W^{\diamond}, \_)$. Definition 9 now implies that $f(T) \overset{\mathsf{wr}, x}{\dashrightarrow} f(S)$. If $\lambda = (\mathsf{wr}, x)$ or $\lambda = (\mathsf{ww}, x)$ the argument is analogous to the previous one, so we omit the discussion for these cases.

Let us prove part (ii). Suppose that for some $I, J \in \mathcal{I}$ we have $I \overset{\lambda}{\longleftrightarrow} J$. Definition 9 ensures that $\lambda = (\mathsf{ww}, x)$ for some $x \in \mathsf{Obj}$. Pick a $T \in f^{-1}(I)$ and a $S \in f^{-1}(J)$, we have to explain why either

$$T \xrightarrow{\mathsf{ww}, x} S \quad \text{or} \quad S \xrightarrow{\mathsf{ww}, x} T \tag{9}$$

First observe that Definition 9 ensures that $x \in W_I^{\square} \cap W_J^{\square}$, where $\mathsf{rwsets}(f(T)) = (\_, \_, W_I^{\square})$ and $\mathsf{rwsets}(f(S)) = (\_, \_, W_J^{\square})$. As $T \Vdash I$ and $S \Vdash J$, we have that $T \vdash \mathtt{write}(x, \_)$ and $S \vdash \mathtt{write}(x, \_)$. Since the order $X.<_{\mathsf{ar}}$ is total by definition, Definition 2(2) imply the required (9) above.

◀

▶ **Lemma 12.** $\forall(\mathsf{DDG}(X), \mathsf{level}). \forall(\mathsf{SDG}(\mathcal{I}), \mathsf{level}_{\mathrm{S}}). (\mathsf{DDG}(X), \mathsf{level}) \lhd (\mathsf{SDG}(\mathcal{I}), \mathsf{level}_{\mathrm{S}})$ *we have that (i) for wm* $\in \{\mathsf{CC}, \mathsf{PC}\}$ *if* $\mathsf{DDG}(X)$ *contains a wm-critical cycle then* $\mathsf{SDG}(X)$ *contains a wm-critical cycle; (ii) for wm* $\in \{\mathsf{PSI}, \mathsf{SI}\}$ *if* $X \models$ CONFLICT *and* $\mathsf{DDG}(X)$ *contains a wm-critical cycle then* $\mathsf{SDG}(X)$ *contains a wm-critical cycle.*

Proof. Fix a pair $(\mathsf{SDG}(X), \mathsf{level}) \lhd (\mathcal{I}, \mathsf{level}_{\mathrm{S}})$ and suppose that $\mathsf{SDG}(X)$ contains a *wm*-critical cycle, say π. We have to prove that $\mathsf{SDG}(\mathcal{I})$ contains a *wm*-critical cycle. Suppose wlog that

$$\pi = T_0 \xrightarrow{\lambda_0} T_1 \xrightarrow{\lambda_1} \dots \xrightarrow{\lambda_{n-1}} T_n = T_0.$$

We use π to construct a cycle $\pi' \in \mathsf{SDG}(\mathcal{I})$ that is $\mathsf{level}_{\mathrm{S}}$ *wm*-critical. The definition of $\lhd$ ensures that there exists a total function h such that

(i) $T \overset{\lambda}{\dashrightarrow} S \in \mathsf{DDG}(X) \implies h(T) \overset{\lambda}{\dashrightarrow} h(S) \in \mathsf{SDG}(\mathcal{I})$;
(ii) $I \overset{\lambda}{\longleftrightarrow} J \in \mathsf{SDG}(\mathcal{I}) \implies$
$\forall T \in h^{-1}(I). \forall S \in h^{-1}(J). T \xrightarrow{\lambda} S \in \mathsf{DDG}(X) \vee S \xrightarrow{\lambda} T \in \mathsf{DDG}(X)$;
(iii) $\mathsf{level}(T) = \mathsf{level}_{\mathrm{S}}(h(T))$.

As h is total, the existence of π and (i) let us show that the graph $\mathsf{SDG}(\mathcal{I})$ contains the following path

$$\pi' = h(T_0) \overset{\lambda_0}{\dashrightarrow} h(T_1) \overset{\lambda_1}{\dashrightarrow} \dots \overset{\lambda_{n-1}}{\dashrightarrow} h(T_n)$$

Since $T_0 = T_n$ and h is a function, we have that $h(T_0) = h(T_n)$, and thus, π' is a cycle.

Before proceeding with the main argument, we show an ancillary fact. Let $I_i = h(T_i)$. For every $i \in [1; n]$

$$T_i \xrightarrow{\lambda} T_{i+1} \text{ is unprotected} \implies I_i \overset{\lambda}{\dashrightarrow} I_{i+1} \text{ is unprotected} \tag{10}$$

This is true because either $\mathsf{level}(T_i) \neq \mathsf{SER}$ or $\mathsf{level}(T_{i+1}) \neq \mathsf{SER}$, thus (iii) above implies that $\mathsf{level}_{\mathrm{S}}(I_i) \neq \mathsf{SER}$ or $\mathsf{level}_{\mathrm{S}}(I_{i+1}) \neq \mathsf{SER}$.

Now we proceed with the main argument, which is a case analysis on wm.

Suppose $wm = \mathsf{CC}$. Definition 4 guarantees that π contains two unprotected edges $T_i \xrightarrow{\mathsf{rw},x} T_{i+1}$ and $T_j \xrightarrow{\lambda} T_{j+1}$ with $\lambda \in \{(\mathsf{rw},\_), (\mathsf{ww},\_)\}$. It follows that $lambda_i = (\mathsf{rw}, x)$ and that $\lambda_j \in (\mathsf{rw},\_), (\mathsf{ww},\_)$, so thanks to (10) we know that the edges that the edges $I_i \overset{\mathsf{rw},\,x}{\dashrightarrow} I_{i+1}$ and $I_j \xrightarrow{\lambda} I_{j+1}$ are unprotected. As π' satisfies the conditions required by Definition 11, and it is CC-critical.

Suppose $wm = \mathsf{PC}$. Definition 4 ensures that π contains two adjacent unprotected edges. Let $T_i \xrightarrow{\mathsf{rw},x} T_{i+1}$ and $T_j \xrightarrow{\lambda} T_{j+1}$ be these edges. We have $i + 1 = j$, thus the edges $I_i \overset{\mathsf{rw},\,x}{\dashrightarrow} I_{i+1}$ and $I_j \xrightarrow{\lambda} I_{j+1}$ are adjacent, and they are unprotected. This means that π' is PC-critical.

Now let us suppose that $X \models$ Conflict. We prove the last two parts of the lemma. Suppose $wm = \mathsf{PSI}$. To show that π' is critical, Definition 11 requires us to prove the following facts,

(a) π' contains at least two unprotected critical rw edges
(b) for every $I_i \overset{\mathsf{rw},\,x}{\dashrightarrow} I_{i+1}$, $I_j \overset{\mathsf{rw},\,y}{\dashrightarrow} I_{j+1} \in \pi'$, if $i \neq j$, then $x \neq y$
and the assumption that π be PSI-critical, together with Definition 4 ensure that

1. π contains at least two unprotected rw edges; and
2. for every $T_i \xrightarrow{\mathsf{rw},\,x} T_{i+1}$, $T_j \xrightarrow{\mathsf{rw},\,y} T_{j+1} \in \pi$, if $i \neq j$, then $x \neq y$.

We prove first (b), for the argument is simple. Fix two edges $I_i \xrightarrow{\mathsf{rw},x} I_{i+1}$ and $I_j \xrightarrow{\mathsf{rw},y} I_{j+1}$ with $i \neq j$. The construction of π' ensures that $T_i \xrightarrow{\mathsf{rw},x} T_{i+1} \in \pi$ and $T_j \xrightarrow{\mathsf{rw},y} T_{j+1} \in \pi$. As $i \neq j$ and π is critical, part (2) above ensures the required $x \neq y$.

Now we prove (a). Part (1) ensures that there exists $i, j \in [0; n]$ such that $i \neq j$, and that

$$T_i \xrightarrow{\mathsf{rw},x} T_{i+1},\ T_j \xrightarrow{\mathsf{rw},y} T_{j+1} \in \pi,$$

with these edges being critical and unprotected, and yielding the edges

$$I_i \overset{\mathsf{rw},\,x}{\dashrightarrow} I_{i+1},\ I_j \overset{\mathsf{rw},\,y}{\dashrightarrow} I_{j+1} \in \pi'.$$

(10) ensures that both edges are unprotected, and now we show that the edge $I_i \overset{\mathsf{rw},\,y}{\dashrightarrow} I_{i+1}$ is critical in π'; the same argument applies also to the other edge.

Fix a pair I_l, I_m of different vertices on π' such that for some $t, t' \in D^\star$ we have $I_l \overset{t}{\dashrightarrow}{}^\star I_i \in \pi'$ and $I_{i+1} \overset{t'}{\dashrightarrow}{}^\star I_m \in \pi'$. We have to show that $\neg(I_l \xleftrightarrow{\mathsf{ww},x} I_m)$ for every x. Let $\mathsf{rwsets}(I_l) = (\_,\_,W_l^\square)$ and $\mathsf{rwsets}(I_m) = (\_,\_,W_m^\square)$. Definition 9 requires us to show that $W_l^\square \cap W_m^\square = \emptyset$. Thanks to (ii) for this it suffices to prove

$$\forall x, y.\, T_l \vdash \mathtt{write}(x,\_) \wedge T_m \vdash \mathtt{write}(y,\_) \implies x \neq y,$$

which we show now. The construction of π' ensures that

$$T_l \xrightarrow{t}{}^\star T_i \xrightarrow{\mathsf{rw},x} T_{i+1} \xrightarrow{t'}{}^\star T_m \in \pi.$$

and Definition 4 ensures that π is chord-free and rw-minimal. Hence,

$$\forall x.\, \neg(T_l \xrightarrow{\mathsf{ww},x} T_m). \tag{11}$$

The assumption that $X \models$ Conflict and Lemma 8 ensure that $\mathsf{DDG}(X)$ satisfies the following property

$$\forall x.\, \neg(T_m \xrightarrow{\mathsf{ww},x} T_l). \tag{12}$$

Now thanks to $X \models$ CONFLICT, from Definition 2, (11) and (12) we get that whenever $T_l \vdash \mathtt{write}(x, \_)$ and $T_m \vdash \mathtt{write}(y, \_)$, we have $x \neq y$, as desired.

Suppose $wm = \mathsf{SI}$. The argument in this case is analogous to the one for PSI, and exploits the additional condition given by Definition 4 that the two critical and unprotected rw edges at hand are adjacent. ◀

▶ **Theorem 13**. *For every (H, level), $(\mathcal{I}, \mathsf{level}_\mathrm{S})$ and wm, if $\mathsf{SDG}(\mathcal{I})$ has no wm-critical cycles and $H \in \mathsf{hist}(wm)$ then $(H, \mathsf{level}) \Vdash (\mathcal{I}, \mathsf{level}_\mathrm{S})$ implies $H \in \mathsf{hist}(\mathsf{SER})$.*

Proof. Fix a wm, an annotated history (H, level) and a pair a pair $(\mathcal{I}, \mathsf{level}_\mathrm{S})$ such that $(H, \mathsf{level}) \Vdash (\mathcal{I}, \mathsf{level}_\mathrm{S})$. If $(H, \mathsf{level}) \in \mathsf{hist}(wm)$ there exists an execution X such that $(H, \mathsf{level}) = (X.H, X.\mathsf{level})$ and that $X \models wm$. Lemma 10 now ensures that

$$(\mathsf{DDG}(X), \mathsf{level}) \lhd (\mathsf{SDG}(\mathcal{I}), \mathsf{level}_\mathrm{S}),$$

and Lemma 12 guarantees that if $\mathsf{DDG}(X)$ contains a wm-critical cycle then $\mathsf{SDG}(\mathcal{I})$ contains a wm-critical cycle. But this would contradict the hypothesis, thus $\mathsf{DDG}(X)$ contains no wm-critical cycle. Theorem 5 now ensures that $\mathsf{DDG}(X)$ is acyclic, thus Lemma 3 implies that $X.H \in \mathsf{hist}(\mathsf{SER})$, as required. ◀