A Compoundable Specification of the Snapshot Isolation Level

Josep M. Bernabé-Gisbert, Francesc D. Muñoz-Escoí

Universitat Politècnica de València, 46022 València (Spain)
Tel.: +34 963877069
Fax: +34 963877239

{jbgisber,fmunyoz}@iti.upv.es

Technical Report ITI-SIDI-2012/007

A Compoundable Specification of the Snapshot Isolation Level

Josep M. Bernabé-Gisbert, Francesc D. Muñoz-Escoí

Universitat Politècnica de València, 46022 València (Spain)
Tel.: +34 963877069
Fax: +34 963877239

Technical Report ITI-SIDI-2012/007

e-mail: {jbgisber,fmunyoz}@iti.upv.es

June 12, 2012

Abstract

Most database management systems support several isolation levels, including Serializable, Snapshot Isolation, and Read Committed. Serializable is used when transactions have strong isolation needs. Read Committed provides a very relaxed isolation and, thus, it is used only when applications are able to deal with the isolation anomalies that such level allows. Snapshot Isolation (SI) is widely used since it provides an almost serializable isolation with improved performance, particularly in read-intensive workloads since it never blocks read accesses nor aborts read-only transactions. Unfortunately, few theoretical models indicate when the execution of a set of transactions follows each isolation level conditions when different isolation levels are requested by different transactions. They are either ambiguous or oriented to concrete concurrency control mechanisms like locks. An important exception is the model proposed by Atul Adya. Such specification proposes Mixed Serialization Graphs (MSG) as a tool for representing executions that involve multiple isolation levels. Adya's MSGs are both well-defined and mechanism-independent, although they do not consider SI since its specification is based on some transaction dependences that do not arise in any other isolation level. This paper provides an alternative SI specification that can be embedded in MSGs.

1 Introduction

Database management systems (DBMS) can concurrently execute transactions requesting different isolation levels. Serializable, Snapshot Isolation, Repeatable Read, Read Committed and Read Uncommitted are some of the isolation levels [12] being regularly supported. The advantage of managing several isolation levels is performance [10]. With Serializable, the strictest isolation level, the execution of a set of transactions is equivalent to a serial execution of the same set even if they are executed concurrently [7]. However, Serializable isolation increases the number of blocked transactions, transaction aborts and response time, reducing the degree of concurrency and the overall performance. Relaxed isolation levels increase concurrency at the cost of allowing some types of isolation anomalies. For example, with Read Committed, the default isolation level in some DBMSs (e.g., Microsoft SQL Server 2008 R2 [18], Oracle Database 11g [17], PostgreSQL 9.1.3 [19]), a transaction may get different values in two consecutive reads on the same data since it allows other concurrent transactions to update data between both reads.

Weak isolation levels should be used only when their unmanaged anomalies cannot appear during the execution of a given transaction or are tolerated by applications and users. Furthermore, databases may be accessed by several applications and, thus, it is usual for a DBMS to manage the execution of concurrent transactions with different isolation needs. The coexistence of those transactions improves the overall

system performance and compromises isolation guarantees only when they are tolerated by the application tier.

There are several mechanisms to manage concurrency in the presence of several isolation levels. They are based on locking, multi-versioning or both. Only a few models can formally describe how those systems should behave to manage isolation correctly and most of them assume locking as their concurrency control mechanism [3]. The model developed by Adya [1] is an exception. Adya's specification has been widely followed in recent years, because of its generalized isolation characterization [13, 8, 14, 16] and its clear and mechanism-independent specification for the snapshot isolation level [10, 15]. His specification uses variations of serialization graphs to define the isolation levels with independence on concrete concurrency control mechanisms. Serialization graphs were originally introduced in [7] to define the Serializable isolation but Adya extended them to define other isolation levels as well. Given an execution, in a Direct Serialization Graph (DSG) the nodes represent transactions and the directed edges represent dependences between them. The execution of a set of transactions guarantees a given isolation level if the associated DSG keeps some properties, usually related to the absence of cycles. When transactions with different isolation levels are executed, Adya introduced a variation of DSGs, named Mixed Serialization Graphs (MSG). An MSG is like the DSG but including only obligatory edges. An edge is obligatory if it is representative for the isolation levels of at least one of the transactions directly involved in the dependency.

Unfortunately, MSGs do not consider transactions with the Snapshot Isolation (SI) level. The reason is that Adya introduced another variant of DSG, named Start-ordered Serialization Graph or SSG, to define that level. Since MSGs are based on DSGs, they only consider isolation levels defined with DSGs and that excludes SI. In this paper we present an alternative SI definition, very similar to Adya's definition but based on DSGs. Our specification can be supported by MSGs. The resulting MSGs are able to model executions that encompass transactions using the isolation levels generally available in current DBMSs and they also provide a basis to reason about the correctness of database replication protocols supporting multiple isolation levels [5]. Regarding the latter, the designers of those replication protocols should take care that the validation rules being used for deciding whether a transaction will be committed or not, considering any concurrent transactions, ensure that the resulting transaction dependences are only those admitted in a valid MSG.

The rest of this paper is structured as follows. Related work is outlined in Section 2. Section 3 describes the system model. Section 4 summarizes the isolation level specification given in [1, 2]. Section 5 provides an alternate specification for the *snapshot* isolation level, integratable in MSGs. Finally, Section 6 concludes the paper.

2 Related Work

Correctness in executions where multiple concurrent transactions are involved has regularly assumed serializable isolation [7]. Other isolation levels introduce the risk of anomalies [3], generating several kinds of inconsistent results. Despite this, relational DBMSs support several standard isolation levels, and most of them use the *Read Committed* level by default. A relaxed isolation is able to enhance the application performance [6, 10]. As a result, the programmer should carefully select which is the most appropriate isolation level for each one of the application transactions, according to the application requirements and semantics. Indeed, Fekete [9] showed that, following certain rules, a careful mixing of serializable and SI transactions is able to generate conflict serializable executions; i.e., those executions are able to avoid all anomalies. Moreover, that work [9] also proposed as a further line of investigation to find possible rules for mixing transactions using read-committed, SI and serializable isolations in order to still obtain conflict serializable executions. This shows that the usage of multiple isolation levels in real applications is convenient from a practical point of view and that still demands some support from a theoretical perspective.

The proposal of the semantic correctness concept in [6] has a similar aim: to set the rules that ensure a correct execution of multiple transactions that use any of the standard isolation levels in relational databases. However, in this case the semantic correctness being proposed is slightly weaker than serializability. Despite this, the rules for managing SI are only given for conventional databases (i.e., for those that do not use predicates [6]), but not for relational ones. As a result, SI is not fully supported in [6], despite admitting that executions with multiple isolation levels are convenient when throughput and response time

should be optimized.

The focus of the current paper is not placed on obtaining conflict serializable executions when multiple isolation levels are mixed, but only in adequately representing the resulting dependences in a mixing graph. The usefulness of that representation resides in guiding the designers and developers of database replication protocols on the set of rules that should be considered in order to check which conflicts should be allowed and avoided (against other committed transactions, depending on their isolation level) when transactions reach their commitment point. As a result, we are mainly focused on replication transparency, one of the aims of any distributed system [20]: i.e., in providing the same support in a replicated environment than in a single-machine one. This demands database replication protocols managing multiple isolation levels, since regular DBMSs always provide such management.

To this end, as it has been already said in the introduction, Adya [1] provided a solid model for supporting such mixed executions in his MSGs. Unfortunately, MSGs did not consider SI since the latter demands another kind of graph: the SSG. In our previous work [4] we tried to complement MSGs demanding two additional rules for the SI transactions involved in a mixed execution. This allowed to include SI transactions in those "extended MSGs", but not in a regular way. In fact, the result was not a regular MSG nor any other kind of plain graph since it should be complemented by two building rules maintaining an information similar to that of the start dependences that originated the SSGs. The model being proposed in the current paper solves this issue.

3 System Model

This section presents some definitions needed in the rest of the paper.

3.1 Databases and transactions

A database is a set of items that can be read and written. Updates, inserts and deletes are all treated as writes. Clients (usually applications) read and write database items through transactions. A transaction is a sequence of read and write operations plus an initial start operation and a final commit or abort operation. If the transaction is committed all its writes are persisted in the database. If it is aborted all writes are rolled back.

Operation $w_i(x_i)$ represents transaction T_i's write on item x, being x_i the value written. Operation $r_i(x_j)$ represents T_i's read of the value x_j written by transaction T_j. T_i's start, commit and abort operations are represented as s_i, c_i and a_i. Note that c_i and a_i are mutually exclusive, i.e., transactions either commit or abort. If a transaction performs several writes on item x, $w_i(x_{i.n})$ represents the n-th write on item x performed by T_i. If no suffix is present, x_i represents the last value established by T_i. Operation $r_i(x_{j.n})$ indicates a T_j's read of the n-th T_i's write on x and $r_i(x_j)$ T_i's read of T_j's last write on x. Finally, x_0 is the initial value of an item x and o_i represents any operation performed by T_i.

A transaction T_i is defined in the following way:

Definition 1 (Transaction)**.** *A transaction T_i is a totally ordered set of operations with a binary relation $<$ where:*

- $T_i \subseteq \{r_i(x_j), w_i(x_i) | x \text{ is a data item}\} \cup \{s_i, a_i, c_i\}$.
- $s_i \in T_i$.
- $c_i \in T_i$ *iff* $a_i \notin T_i$.
- *For any T_i's operation o_i, if $o_i \neq s_i$ then $s_i < o_i$.*
- *If $c_i \in T_i$ then, for any operation $o_i \neq c_i$, $o_i < c_i$.*
- *If $a_i \in T_i$ then, for any operation $o_i \neq a_i$, $o_i < a_i$.*
- *For any two T_i's operations o_1 and o_2, $o_1 < o_2$ or $o_2 < o_1$.*

3.2 Histories

When a set of transactions is executed in the system, the operation execution order is determined by a system scheduler. A history represents how transactions have been ordered during the execution. Given two operations o_1 and o_2, $o_1 <_H o_2$ in a history H if they have been executed in that order and either belong to the same transaction or are conflictive. Operations o_1 and o_2 conflict if they operate over the same item and at least one of them is a write. Thus, two read operations of distinct transactions never conflict and are not directly ordered in H. Formally:

Definition 2 (History)**.** *Given a set of transactions $\mathscr{T} = \{T_1, ..., T_n\}$, a history H is a partially ordered set of the operations in $\mathscr{T}$'s transactions with a binary relation $<_H$ where:*

- *For any transaction $T_i \in \mathscr{T}$ and any operation $o_i \in T_i$, $o_i \in H$.*
- *For any transaction $T_i \in \mathscr{T}$ and any two operations $o_1, o_2 \in T_i$, if $o_1 < o_2 \in T_i$ then $o_1 <_H o_2 \in H$.*
- *If $r_i(x_j) \in H$ then $w_j(x_j) \in H$ and $w_j(x_j) <_H r_i(x_j) \in H$.*
- *For any two conflicting operations $o_1, o_2 \in H$, $o_1 <_H o_2 \in H$ or $o_2 <_H o_1 \in H$.*

3.3 Time-precedes order

The scheduler assigns transaction start and end points. They represent when transactions start and finish and determine which committed database state is observed by every transaction when it is started. That order is called a *time-precedes order* [1]. Formally:

Definition 3 (Time-precedes order)**.** *Given a history H and E the set of start and commit operations of transactions committed in H, a time-precedes order $<_t$ is a partial order on E such that:*

- *For any transaction T_i committed in H, $s_i <_t c_i$.*
- *Given T_i, T_j transactions committed in H, $c_i <_t s_j$ or $s_j <_t c_i$.*

Two transactions T_i and T_j are concurrent if $s_i <_t c_j$ and $s_j <_t c_i$.

4 Isolation levels

Ensuring a strict isolation is costly and, depending on the concrete mechanism being used, this implicates many blocked transactions and/or many aborts. To improve performance, commercial DBMSs allow transactions to be executed with weaker isolation levels at the cost of allowing certain types of interferences or phenomena, which must be either managed by the application tier or accepted by the user. An example is the phenomenon known as Write Skew [3]. Assume a database with two items x and y and an integrity constraint requiring that $x + y > 0$. If two transactions T_i and T_j concurrently read $x = 50$ and $y = 50$ and later are allowed to write $x = -10$ (T_i) and $y = 0$ (T_j), both will think that the integrity constraint is preserved but it is actually violated in the final state. If a transaction is executed with an isolation level which does not prevent this phenomenon we must be sure that such scenario is managed by the application logic avoiding that kind of concurrent transactions.

Snapshot Isolation is an isolation level widely used in DBMSs. It provides almost the same isolation guarantees than Serializable but it never blocks read operations. As a result, read-only transactions can be executed without being blocked or aborted.

4.1 Adya's isolation level definitions

Several isolation level specifications have been given in the literature [12, 3, 1, 2]. They identify possible phenomena that may appear when transactions are executed concurrently and categorize isolation levels depending on which of those phenomena are forbidden. The ANSI/INCITS specification [12] is widely

accepted but, as Berenson et al. [3] showed up, it is ambiguous. Berenson et al. [3] refined ANSI definitions and extended the classification with new phenomena and isolation level definitions. Actually, they suggested one of the first definitions of Snapshot Isolation, supported at that moment by some commercial DBMS as Serializable due to a loose interpretation of ANSI phenomena. They proved that SI allows some non-serializable executions. Indeed, other papers [11] showed up that there were other anomalies in SI histories. However, Berenson's specification focuses on lock-based concurrency control, ignoring other mechanisms like multi-versioning, widely used to provide Snapshot Isolation.

Due to that fact, Adya [1] presented an alternative specification that is independent of concrete concurrency control mechanisms. Adya used a variation of Bernstein's serialization graphs to represent histories as graphs showing dependencies among transactions. Phenomena are defined as properties in those graphs.

Definition 4 (Direct Serialization Graph)**.** *Given a history H, $DSG(H)$ is a directed graph containing a vertex per committed transaction in H and a directed edge from T_i to T_j if one of the following dependencies appears:*

- *T_j directly read-depends on T_i, denoted as $T_i \xrightarrow{wr} T_j$, if $r_j(x_i) \in H$.*
- *T_j directly write-depends on T_i, denoted as $T_i \xrightarrow{ww} T_j$, if $w_i(x_i) <_H w_j(x_j) \in H$ and it does not exist any other operation $w_k(x_k)$ such that $w_i(x_i) <_H w_k(x_k) <_H w_j(x_j) \in H$.*
- *T_j directly anti-depends on T_i, denoted as $T_i \xrightarrow{rw} T_j$, if $r_i(x_m) <_H w_j(x_j) \in H$ and it does not exist any other operation $w_k(x_k)$ such that $r_i(x_m) <_H w_k(x_k) <_H w_j(x_j) \in H$.*

We say that T_j directly depends or depends on T_i if T_j directly read- or write-depends on T_i. We also say that T_j anti-depends on T_i if it directly anti-depends on T_i.

As an example of DSG, given a history $H = w_0(x_0)w_0(y_0)w_0(z_0)c_0r_i(x_0)w_i(x_i)r_i(y_0)c_iw_j(y_j)w_j(x_j)c_j$ (this is the flatten representation of H but remember that a history is a partial order and not a total order), the associated $DSG(H)$ is:

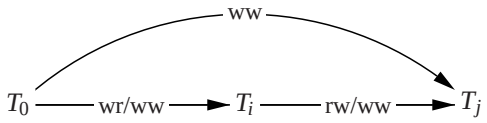

Figure 1: DSG of H_1

Adya used DSG to define a set of possible isolation phenomena. The main ones are the following:

- **G0: Write Cycles**: a history H exhibits phenomenon G0 if $DSG(H)$ contains a directed cycle composed only by write-dependency edges.
- **G1a: Aborted Reads**: a history H exhibits phenomenon G1a if it contains an aborted transaction T_i and a committed transaction T_j such that $w_i(x_{i.m}) <_H r_j(x_{i.m}) \in H$.
- **G1b: Intermediate Reads**: a history H exhibits phenomenon G1b if a transaction T_i reads in H a value written by T_j which is not the last write of T_j over the item. Formally, $w_i(x_{i.m}) <_H r_j(x_{i.m}) <_H w_i(x_{i.n}) \in H$ and $c_j \in H$.
- **G1c: Circular Information Flow**: a history H exhibits phenomenon G1c if $DSG(H)$ contains a directed cycle composed only by dependency edges.
- **G2: Anti-dependency Cycles**: a history H exhibits phenomenon G2 if $DSG(H)$ contains a directed cycle containing at least one anti-dependency edge.

Based on the previous phenomena, the following isolation levels are defined:

- **PL-1**: it forbids phenomenon G0 and provides a generalized specification for Read Uncommitted.

- **PL-2**: it forbids phenomena G0, G1a, G1b and G1c and provides a generalized specification for Read Committed.
- **PL-3**: it forbids phenomena G0, G1a, G1b, G1c and G2 and provides a generalized specification for Serializable.

Instead of focusing on what should be observed in every transaction execution to determine if its isolation requirements have been ensured, Adya's definitions indicate what should happen in an entire history to guarantee a given isolation level to all comitted transactions. Thus, if PL-1, PL-2 and PL-3 transactions are executed, we do not know if isolation requirements have been ensured to every transaction but which isolation level is ensured to the whole transaction set execution represented by H. To fill that void, Adya suggested a variation of serialization graphs named Mixed Serialization Graphs (MSG). Given a history H and its $DSG(H)$, $MSG(H)$ has all $DSG(H)$ nodes but only those edges representing obligatory dependencies for one of the involved transactions isolation levels. The overall execution is correct if $MSG(H)$ does not have cycles and does not show G1a and G1b phenomena for PL-2 and PL-3 transactions. The obligatory dependencies are the following:

- All direct write-dependencies.
- Direct read-dependencies ending in PL-2 and PL-3 transactions.
- Direct anti-dependencies starting from a PL-3 transaction.

Unfortunately, MSGs do not consider transactions requesting the SI level. The reason is that the SI specification proposed by Adya, named PL-SI, is not based on DSGs but on another variation named Start-dependency Serialization Graphs (SSGs). Given a history H and a time-precedes order $<_t$, $SSG(H,<_t)$ has all $DSG(H)$ nodes and edges plus start-dependency edges:

- T_j **start-depends** on T_i, denoted as $T_i \xrightarrow{s} T_j$, if $c_i <_t s_j$.

SSGs consider new phenomena:

- **G-SIa: Interference**: a history H and a time-precedes order $<_t$ exhibit the phenomenon G-SIa if $T_i \xrightarrow{ww} T_j \in SSG(H,<_t)$ or $T_i \xrightarrow{wr} T_j \in SSG(H,<_t)$ but $T_i \xrightarrow{s} T_j \notin SSG(H,<_t)$.
- **G-SIb: Missed Effects**: a history H and a time-precedes order $<_t$ exhibit the phenomenon G-SIb if $SSG(H,<_t)$ contains a directed cycle with exactly one direct anti-dependency edge.

PL-SI isolation level forbids phenomena G0, G1a, G1b, G1c, G-SIa and G-SIb.

4.2 An alternative definition for G-SIb

PL-SI actually forbids more cycles than those explicitly forbidden by G0, G1c and G-SIb. Thus, we provide an alternative definition which explicitly excludes all graphs representing non PL-SI executions:

Definition 5 (New G-SIb: Missed Effects)**.** *A history H and a time-precedes order $<_t$ exhibit the phenomenon New G-SIb if $SSG(H,<_t)$ contains a directed cycle with at least one direct anti-dependency edge but without two consecutive direct anti-dependency edges.*

Lemma 1 proves that both G-SIb definitions can be indistinctly used in PL-SI definition.

Lemma 1 (G-SIb and New G-SIb are equivalent)**.** *Given a history H and a times precedes order $<_t$, $SSG(H,<_t)$ forbids G0, G1a, G1b, G1c, G-SIa and GSI-b phenomena iff $SSG(H,<_t)$ forbids G0, G1a, G1b, G1c, G-SIa and New G-SIb phenomena.*

Proof. New G-SIb is less restrictive than the original G-SIb and, thus, when New G-SIb is proscribed, it is admitting less histories than the original definition. Conversely, we prove that any history H and time-precedes order $<_t$ proscribing G0, G1a, G1b, G1c, G-SIa and G-SIb proscribes also New G-SIb. This implies that both conditions sets are equivalent. By absurd reduction, we assume $SSG(H,<_t)$ is PL-SI

and it has a cycle with anti-dependency edges but without two consecutive anti-dependency edges. Since $SSG(H, <_t)$ is PL-SI, it has not any cycle with a single anti-dependency edge and, thus, the cycle has at least two of those edges. Assume that one of them goes from T_i to T_j, $T_i \xrightarrow{rw} T_j$. Thus, $s_i <_t c_j$ because, otherwise, $c_j <_t s_i$ and there is a start-dependency edge from T_j to T_i which closes a cycle with a single anti-dependency edge which contradicts the initial assumption. Since there are not two consecutive anti-dependency edges, there must be an edge from another node T_k to T_i and another one from T_j to T_l, both dependency or start-dependency edges. Thus, by G-SIa and start-dependency definitions, $c_k <_t s_i$ and $c_j <_t s_l$. Since $s_i <_t c_j$, then $c_k <_t s_l$ and this implies that $T_k \xrightarrow{s} T_l \in SSG(H, <_t)$. Consequently, there is a shorter cycle without $T_i \xrightarrow{rw} T_j$ anti-dependency edge. We can iteratively apply the same criterion until getting a cycle with a single anti-dependency edge which contradicts the initial assumption saying that H and $<_t$ avoid the original G-SIb phenomenon. □

5 Alternative definition of Snapshot Isolation

A history H is PL-SI if it exists a scheduler $S(E, <_t)$ such that E is the set of start and commit events of committed transactions in H, $<_t$ is a time-precedes order on E and $SSG(H, <_t)$ does not show G0, G1a, G1b, G1c, G-SIa and G-SIb phenomena. This section introduces PL-SI', an alternative definition exclusively based on DSG dependency edges. We also prove that for any PL-SI' history H there exists at least one scheduler $S(E, <_t)$ such that $SSG(H, <_t)$ is PL-SI.

5.1 PL-SI': an alternative definition of PL-SI

With Snapshot Isolation (SI) a transaction is executed over a committed state of the database or snapshot. If a transaction T_i reads or overwrites a value x_j then transaction T_j is in T_i's snapshot. If T_i reads a value overwritten by another transaction T_k then T_k is not in T_i's snapshot. If T_j is in T_i's snapshot and T_l is in T_j's snapshot then T_l is also in T_i's snapshot. Thus, a transaction is *in a* snapshot s if its updates can be read from s or if they were accessible at a previous committed state and have been later overwritten by transactions also in s.

In a SI history H, T_i cannot be involved in a cycle composed only by dependency edges (G1c phenomenon). T_i cannot be part of the snapshot it observes while it is being executed. Cycles with only one anti-dependency edge are neither possible. If we assume that such cycle exists and the anti-dependency goes from T_i to T_j then there is a path from T_j to T_i in $DSG(H)$ composed only by dependency edges. Thus, T_j is and is not in T_i's snapshot at the same time, which is a contradiction. As in Section 4.2, that can be extended to any cycle with anti-dependency edges as soon as two or more of them do not appear consecutively[1].

Consequently, G-SIb can be redefined in the following way:

Definition 6 (G-SIb': Missed Effects)**.** *A history H exhibits the phenomenon G-SIb' if $DSG(H)$ contains a directed cycle with at least one direct anti-dependency edge but without two consecutive direct anti-dependency edges.*

Thus, PL-SI' can be defined in the following way:

Definition 7 (PL-SI')**.** *A history H is PL-SI' if it forbids G0, G1a, G1b, G1c and G-SIb'.*

5.2 PL-SI' and PL-SI equivalence

PL-SI' and PL-SI do not represent actually the same, basically because PL-SI' is based on the dependencies in H but PL-SI contemplates also the dependencies among the start and commit operations in E. However,

[1] Imagine a cycle with two or more non-consecutive anti-dependency edges. This cycle should have at least four nodes. If T_1, T_2, T_3 and T_4 are four consecutive nodes in the cycle and there is an anti-dependency edge from T_2 to T_3 then the edges from T_1 to T_2 and from T_3 to T_4 are dependency edges. Thus, T_1 is in T_2's snapshot and T_3 is in T_4's snapshot but is not in T_2's snapshot. Consequently, T_2 observes a committed state previous to T_4 snapshot. Since T_1 is in T_2's snapshot then T_1 is also in T_4's snapshot. If that is iteratively applied to the rest of anti-dependency edges we will finally reach to a contradiction like T_1 is in T_1's snapshot (note that this means that the items written by T_1 had been committed before T_1 started, and this is impossible).

it can be said that PL-SI' histories are also PL-SI if they represent a correct PL-SI execution. In other words, PL-SI' is equivalent to PL-SI if for any PL-SI' history H exists at least one scheduler $S(E, <_t)$ such that $SSG(H, <_t)$ is PL-SI. In this section we prove that at least one scheduler like this exists and is based on what we call an Extended SI-derived order $<_e$, a specific type of time-precedes order.

The inverse assertion is trivially true. If H and $<_e$ are PL-SI then H is PL-SI' since proscribing G-SIb' is less restrictive than proscribing G-SIb and G-SIa. If there is not a cycle with anti-dependencies but without two consecutive anti-dependencies in $SSG(H, <_e)$ then this cycle obviously does not exist in $DSG(H)$ because both graphs have the same vertices but $DSG(H)$ has only a subset of $SSG(H, <_e)$'s edges.

Both PL-SI and PL-SI' forbid G0, G1a, G1b and G1c phenomena. Then, a PL-SI' history H is PL-SI if $SSG(H, <_e)$ does not show G-SIa and G-SIb and $<_e$ is a times-precedes order. $<_e$ is an extension of a SI-derived order $<_s$. $<_s$ orders starts and commits of committed transactions in H by applying SI rules to the dependencies in $DSG(H)$. In the following we define both, SI-derived order and Extended SI-deriver order.

Definition 8 (SI-derived order)**.** *Given a history H and E the set of start and commit operations of committed transactions in H, a SI-derived order $<_s$ is a partial order on E where:*

a) $c_j <_s s_i$ *if* $T_j \xrightarrow{ww} T_i \in DSG(H)$ *or* $T_j \xrightarrow{wr} T_i \in DSG(H)$.

b) $s_j <_s c_i$ *if* $T_j \xrightarrow{rw} T_i \in DSG(H)$.

c) $s_i <_s c_i$ *if T_i is in H and T_i commits.*

Condition a) avoids G-SIa phenomena and condition b) allows a transaction T_i to miss the effects of transactions only if they do not belong to T_i's snapshot. Condition c) indicates how a transaction start and commit operations are ordered.

Unfortunately, $<_s$ may not order any possible combination of s_i and c_j and, thus, it is not a time-precedes order. For example, it will never order start and commit operations of transactions not connected in $DSG(H)$. The Extended SI derived order $<_e$ we further present is a SI-derived order, a time-precedes order and, most important, applied to a PL-SI' H results in a PL-SI $SSG(H, <_e)$.

Definition 9 (Extended SI-derived order)**.** *Given a history H and E the set of start and commit operations of committed transactions in H, an Extended SI-derived order $<_e$ is a partial order on E where:*

a) $<_e$ *is a SI-derived order on E.*

b) $s_i <_e c_j$ *(alternatively, $c_j <_e s_i$) iff they are ordered that way by the following set-order algorithm. Given $\mathcal{O}$ the set of pairs $\{s_i, c_j\}$ unordered by SI-derived order conditions a), b) and c) and given $<_o$ a total order on $\mathcal{O}$ such that $\{s_i, c_j\} <_o \{s_k, c_l\}$ iff $i < k \vee (i = k \wedge j < l)$, the set-order algorithm orders the elements in $\mathcal{O}$ as follows:*

- *While $\mathcal{O}$ is not empty:*
  - *Take off the minor pair $\{s_i, c_j\}$ from $\mathcal{O}$ ($\nexists \{s_k, c_l\} \in \mathcal{O}$ such that $\{s_k, c_l\} <_o \{s_i, c_j\}$).*
  - *Order its operations as $s_i <_e c_j$.*
  - *Take off from $\mathcal{O}$ any other pair if its operations ordering can be now deduced.*

The idea behind $<_e$ is not to accurately represent how transactions in H have been originally scheduled (actually, this may be not feasible). $<_e$ deduces start-dependencies from dependencies and anti-dependencies in H and suggest an ordering of the rest of transactions starts and commits avoiding any possible violation of PL-SI restrictions. As we prove later in this section, $<_e$ shows that at least the $<_e$ ordering produces PL-SI executions when applied to a PL-SI' history H, which proves that H represents a PL-SI scheduler.

As an example, assume the history $H_1 = w_1(x_1)\ w_2(y_2)\ c_1\ c_2\ w_3(x_3)\ c_3\ w_4(y_4)\ c_4$ and the associated $DSG(H_1)$. The associated $DSG(H_1)$ is shown in Figure 2.

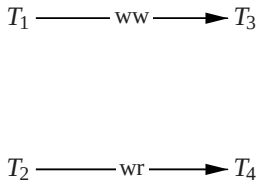

Figure 2: $DSG(H_1)$

$<_e$ orders all commit and start operations of committed transactions in H_1: $\{s_1,c_1\}$, $\{s_1,c_2\}$, $\{s_1,c_3\}$, $\{s_1,c_4\}$, $\{s_2,c_1\}$, $\{s_2,c_2\}$, $\{s_2,c_3\}$, $\{s_2,c_4\}$, $\{s_3,c_1\}$, $\{s_3,c_2\}$, $\{s_3,c_3\}$, $\{s_3,c_4\}$, $\{s_4,c_1\}$, $\{s_4,c_2\}$, $\{s_4,c_3\}$, $\{s_4,c_4\}$. SI-derived order condition a) orders $c_1 <_e s_3$ and $c_2 <_e s_4$. Condition b) cannot be applied in this case and c) orders $s_1 <_e c_1, s_2 <_e c_2, s_3 <_e c_3$ and $s_4 <_e c_4$. Due to the transitivity property of partial order $<_e$, if $s_1 <_e c_1$, $c_1 <_e s_3$ and $s_3 <_e c_3$, then $s_1 <_e c_3$ can be deduced. Similarly, $s_2 <_e c_4$ can also be deduced from conditions a) and c) orderings. The unordered pairs are $\mathscr{O}$=($\{s_1,c_2\}$, $\{s_1,c_4\}$, $\{s_2,c_1\}$, $\{s_2,c_3\}$, $\{s_3,c_2\}$, $\{s_3,c_4\}$, $\{s_4,c_1\}$, $\{s_4,c_3\}$). The set-order algorithm pops the first pair in $\mathscr{O}$ and orders it as $s_1 <_e c_2$. Since $c_2 <_e s_4$ and $s_4 <_e c_4$, $s_1 <_e c_4$ can be deduced and it is also popped from $\mathscr{O}$. Then the algorithm orders $s_2 <_e c_1$, deduces $s_2 <_e c_3$ (since $c_1 <_e s_3$ and $s_3 <_e c_3$) and both pairs are also popped. In the following step $s_3 <_e c_2$ is popped and ordered. Since $c_2 <_e s_4$ and $s_4 <_e c_4$, $s_3 <_e c_4$ is also popped. $c_1 <_e s_4$ can be also popped since $c_1 <_e s_3$, $s_3 <_e c_2$ and $c_2 <_e s_4$. Finally, $s_4 <_e c_3$ is popped and ordered. Figure 3 shows the resulting graph:

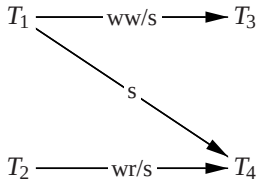

Figure 3: $SSG(H_1)$

As another example, assume a history $H_2 = w_0\ (x_0)\ w_0(y_0)\ c_0\ w_1(x_1)\ c_1\ r_2(y_0)\ c_2\ w_3(x_3)\ w_3(y_3)\ c_3\ w_4(y_4)\ c_4$ which generates the following $DSG(H)$ (T_0 establishes the initial state of the database and, for the sake of simplicity, it is not represented in $DSG(H_2)$). Figure 4 shows the associated $DSG(H_2)$:

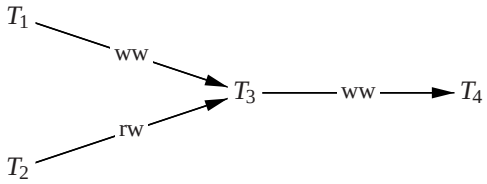

Figure 4: $DSG(H)$

Ordering $<_e$ must order the following start and commit pairs of committed transactions: $\{s_1,c_1\}$, $\{s_1,c_2\}$, $\{s_1,c_3\}$, $\{s_1,c_4\}$, $\{s_2,c_1\}$, $\{s_2,c_2\}$, $\{s_2,c_3\}$, $\{s_2,c_4\}$, $\{s_3,c_1\}$, $\{s_3,c_2\}$, $\{s_3,c_3\}$, $\{s_3,c_4\}$, $\{s_4,c_1\}$, $\{s_4,c_2\}$, $\{s_4,c_3\}$, $\{s_4,c_4\}$. Condition a) orders $c_1 <_e s_3$ and $c_3 <_e s_4$. Condition b) orders $s_2 <_e c_3$. Condition c) orders $s_1 <_e c_1, s_2 <_e c_2, s_3 <_e c_3$ and $s_4 <_e c_4$. From those orderings $s_1 <_e c_3$, $s_1 <_e c_4$, $s_2 <_e c_4$, $s_3 < c_4$ and $c_1 <_e s_4$ can be deduced. Thus, $\mathscr{O} = (\{s_1,c_2\}, \{s_2,c_1\}, \{s_3,c_2\}, \{s_4,c_2\})$. The set-order algorithm will order them as $s_1 <_e c_2$, $s_2 <_e c_1$, $s_3 <_e c_2$ and $s_4 <_e c_2$. Thus, H and $<_e$ represent an execution where T_1, T_3 and T_4 have been scheduled serially and T_2 has been executed concurrently to all of them, which do not violate any PL-SI requisite. The resulting $SSG(H_2, <_T)$ is depicted in Figure 5:

Once defined the Extended SI-derived order $<_e$, we now prove that, applied to any PL-SI' history H, the resulting $SSG(H, <_e)$ is PL-SI. Since PL-SI' avoids G0, G1a, G1b, G1c and G-SIb' phenomena, we

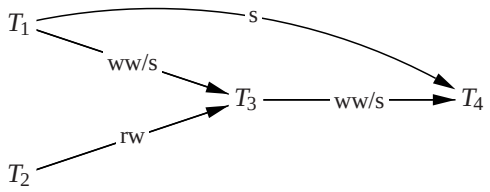

Figure 5: $SSG(H, <_e)$

should prove that that $SSG(H, <_e)$ proscribes G-SIa and G-SIb and that $<_e$ is a time-precedes order. The demostration starts by proving that SI-derived order conditions and the set-order algorithm do not violate G-SIa or G-SIb (Lemmas 2 and 3). Next, Theorem 1 proves that $<_e$ is also a time-precedes order. Finally, all is gathered together in Theorem 2 to prove that $SSG(H, <_e)$ is PL-SI.

Lemma 2 (Correctness of SI-derived order conditions a), b) and c))**.** *Given a PL-SI' history H and a SI-derived order $<_s$, G-SIa and New G-SIb phenomena do not show up in $SSG(H, <_s)$.*

Proof. The condition a) of SI-derived order $<_s$ adds a start-dependency edge $T_i \xrightarrow{s} T_j$ to $SSG(H, <_s)$ for every dependency edge $T_i \xrightarrow{ww/wr} T_j$ in $DSG(H)$. That trivially forbids G-SIa.

Since H is PL-SI', G-SIb' and G1c forbid any cycle in $DSG(H)$ composed by dependency and anti-dependency edges. The only exception are cycles with two or more consecutive anti-dependency edges. By absurd reduction, assume that H is PL-SI' but $SSG(H, <_s)$ shows New G-SIb. Thus, $SSG(H, <_s)$ has a cycle with anti-dependency edges but without two consecutive anti-dependency edges. Since $SSG(H, <_s)$ only differs from $DSG(H)$ in the start-dependency edges and H forbids G-SIb', that cycle contains at least one start-dependency edge. From Lemma 5 (see Appendix), if $T_i \xrightarrow{s} T_j$ in $SSG(H, <_s)$ (i.e., $c_i <_s s_j$) then exists a directed path p in $DSG(H)$ from T_i to T_j such that p starts and ends with dependency edges and does not show two consecutive anti-dependency edges. If we change every start-dependency edge in the cycle by its alternative path we will get a new cycle composed only by dependency and anti-dependency edges and without two consecutive anti-dependency edges. Thus, $DSG(H)$ shows G-SIb' phenomena which contradicts the initial assumption. Thus, $SSG(H)$ forbids New G-SIb. □

The set-order algorithm orders the remaining starts and commits once conditions a), b) and c) are applied. The algorithm itself avoids contradictions since it only orders unordered pairs. Thus, if $c_i <_e s_j$ can be deduced from a), b) and c) or from any other previous ordering applied by the set-order algorithm then this pair will never be in $\mathcal{O}$ and will never be ordered again.

Lemma 3 (Set-order algorithm correctness)**.** *Given a PL-SI' history H and an Extended SI-derived order $<_e$, the start-dependency edges added by the set-order algorithm never produce a New G-SIb cycle in $SSG(H, <_e)$.*

Proof. The set-order algorithm explicitly orders starts before commits only. By absurd reduction, assume that the set-order algorithm adds an ordering $s_i <_e c_j$ and that closes a cycle with anti-dependency edges but without two consecutive anti-dependency edges. Assume the new anti-dependency edge produced by the set-order algorithm goes from vertex T_0 to T_n such that $c_0 <_e s_i$ and $c_j <_e s_n$. Then, there is a path from T_n to T_0 with anti-dependency edges but without two consecutive anti-dependency edges. From Lemma 4 (see Appendix), we can deduce that $s_n <_e c_0$ and, hence, that $c_j <_e s_i$. Since the set-order algorithm only orders start and commit operations if their ordering cannot be deduced from previous orderings, then it will never order $s_i <_e c_j$ which contradicts the initial assumption. Hence, $s_i <_e c_j$ will never close a cycle with anti-dependency edges but without two consecutive anti-dependency edges. □

Finally, we prove that $<_e$ is a time-precedes order and produces a PL-SI $SSG(H, <_e)$ if H is PL-SI'.

Theorem 1. *An Extended SI-derived order $<_e$ is also a time-precedes order.*

Proof. We split the proof in two parts:

- $<_e$ **orders any pair** $\{s_i, c_j\}$ **in** H: some pairs are ordered by SI-derived order conditions a), b) and c). The remaining pairs are ordered by the set-order algorithm in condition d).
- $<_e$ **does not generate contradictions**. In other words, it will never order $s_i <_e c_j$ and $c_j <_e s_i$ at the same time. From Lemma 6 (see Appendix) we know that SI-derived order conditions a), b) and c) will never produce a contradiction and the set-order algorithm only orders unordered pairs and does not introduce contradictions either.

□

Theorem 2. *If H is PL-SI' then $SSG(H, <_e)$ is PL-SI.*

Proof. G0, G1a, G1b, G1c are avoided by definition since H is supposed to be PL-SI'. Thus, we only have to prove that G-SIa and G-SIb are avoided too:

- **G-SIa**: It is trivially avoided by Condition a) of Def. 9.
- **G-SIb**: Lemmas 1, 2 and 3 ensure that Conditions a), b), c) and d) will never produce in $SSG(H, <_e)$ a cycle with exactly one anti-dependency edge.

□

Thus, when time-precedes order $<_e$ is applied to the start and commit operations of committed transactions in a PL-SI' history H, the resulting $SSG(H, <_e)$ is PL-SI. Then, a PL-SI' history H is also PL-SI.

6 Conclusions

Existing commercial DBMSs support several isolation levels to adjust the concurrency control mechanisms to transaction isolation needs. This improves the overall system performance by compromising only the isolation guarantees that can be managed by applications or assumed by the final users. However, today it does not exist a complete and mechanism-independent formal model to determine if a given execution is correctly managing isolation when multiple transactions request different isolation levels.

The closest model is the one introduced by Adya. However, it only supports PL-1, PL-2 and PL-3 isolation levels, equivalent to Read Uncommitted, Read Committed and Serializable but leaves aside the widely used Snapshot Isolation level. SI offers almost the same isolation guarantees than Serializable but shows a better performance in read intensive environments, specially if there are a lot of read-only transactions. SI is not included in Adya's model because it proscribes phenomena based on a time-precedes order and SSG graphs while his model supporting several isolation levels is based on DSGs, which only contemplates dependencies related on conflicts among read and write operations. In this paper we present PL-SI', an alternative definition of SI based on DSGs. We also prove that any PL-SI history H is also PL-SI' and vice versa by showing that it is possible to define a time-precedes order $<_e$ over any PL-SI' history H such that $SSG(H, <_e)$ is PL-SI. Our definition can be included in Adya's MSG theory to prove the correctness of a history when transactions request different isolation levels, including SI.

APPENDIX

Lemma 4 (Start and commit orderings by $<_s$ in PL-SI' histories)**.** *Given a PL-SI history H and a SI-depends order $<_s$, $s_i <_s c_j$ iff*

a) A path p in $DSG(H)$ connects nodes T_i and T_j and

b) p is a directed path from T_i to T_j and

c) p does not contain two consecutive anti-dependency edges.

Proof. a) **If $s_i <_s c_j$ then a), b) and c)**:

- *a) proof.* $<_s$ conditions only order starts and commits of transactions connected by an edge in $DSG(H)$. Thus, $s_i <_s c_j$ if T_i and T_j are connected in $DSG(H)$ by a path of edges.
- *b) and c) proof.* Assume $s_i <_s c_j$ is deduced from path p. We prove by induction over p's length n that p fulfils b) and c) conditions:
  - *Base case ($n = 1$)*: p is composed by a single edge e. Since $s_i <_s c_j$, then e can only be $T_i \xrightarrow{rw} T_j$ (by $<_s$ condition b)) or $T_i \xrightarrow{ww/wr} T_j$ (by the combination of $<_s$ conditions a) and c)). Any other alternative (i.e., an edge from T_j to T_i) will contradict $s_i <_s c_j$ initial assumption. Thus, e fulfils b) and c).
  - *Induction hypothesis ($n < l$)*: if $s_i <_s c_j$ and p length $n < l$ then b) and c) are fulfilled.
  - *Induction step ($n = l$)*: assume $s_i <_s c_j$ and p length is l. Does p fulfil b) and c)? Suppose T_k is the immediate node before T_j in p. Thus, there is a path p' connecting T_i and T_k and a final edge e joining T_k and T_j in $DSG(H)$. p' is $l-1$ length. b) and c) are fulfilled in p only if they are also fulfiled in p' and they are neither violated by e. There are four alternatives:
    - $e = T_k \xrightarrow{ww/wr} T_j$: by $<_s$'s condition a), $c_k <_s s_j$ and, by $<_s$'s condition c), $c_k <_s c_j$. Since $s_i <_s c_j$ is deduced from p, $s_i <_s c_k$ in p'. By the induction hypothesis, p' fulfils b) and c). Since $T_k \xrightarrow{ww/wr} T_j$ goes in the same p' direction and is a dependency edge, p also fulfills b) and c) conditions.
    - $e = T_k \xrightarrow{rw} T_j$: by $<_s$'s condition b), $s_k <_s c_j$. Since $s_i <_s c_j$ is deduced from p, $s_i <_s s_k$ and, by c), $s_i <_s c_k$. By induction hypotesis, p' fulfils b) and c). Imagine $e' = T_{k-1} \longrightarrow T_k$ is the last edge in p'. Since $s_i <_s s_k$, exists a T_{k-1}'s operation o_{k-1} such that $o_{k-1} <_s s_k$ and $s_i <_s o_{k-1}$. o_{k-1} is either s_{k-1} or c_{k-1} since $<_s$ only orders starts and commits. If we observe $<_s$, a start is always ordered with a commit in an edge and, thus, $c_{k-1} <_s s_k$ and e' is a dependency edge. Thus, there are not two consecutive anti-dependency edges and p fulfills b) and c).
    - $e = T_j \xrightarrow{ww/wr} T_k$: in this case p trivially does not fulfil condition b) and, thus, we should prove that $s_i \not<_s c_j$. By $<_s$'s condition a) $c_j <_s s_k$. Even if $s_i <_s s_k$, $s_i <_s c_j$ can not be deduced.
    - $e = T_j \xrightarrow{rw} T_k$: again, p trivially does not fulfil condition b) and, thus, we should prove that $s_i \not<_s c_j$. By $<_s$'s condition b) $s_j <_s c_k$. Even if $s_i <_s c_k$, $s_i <_s c_j$ can not be deduced.

b) **If a), b) and c) then** $s_i <_s c_j$: thus, there is a directed path p from T_i to T_j without two consecutive anti-dependency edges. If p is composed only by dependency edges then trivially $s_i <_s c_j$. Given $p = T_i \xrightarrow{ww/wr} T_0 \xrightarrow{...} T_m \xrightarrow{ww/wr} T_j$, by $<_s$ condition a) $c_i <_s s_0$, ..., $c_m <_s s_j$. Since, by $<_s$ condition c), $s_k <_s c_k$ for any node T_k, $s_i <_s c_i <_s s_0 <_s ... <_s c_m <_s s_j <_s c_j$ and, thus, $s_i <_s c_j$. If there are anti-dependency edges in p, we prove $s_i <_s c_j$ by induction over the number n of anti-dependency edges in p.

- **Base case** $(n = 1)$: there is only one anti-dependency edge e. We differentiate the following cases:
  - e is the only edge in p: by $<_s$ condition b), $s_i <_e c_j$.
  - p is composed by two or more edges and:
    - $e = T_i \xrightarrow{rw} T_k$ is the first edge in p. By $<_s$ condition b), $s_i <_e c_k$. Any other edge $T_n \longrightarrow T_{n+1}$ in p is a dependency edge and, by $<_s$ condition a), $c_n <_e s_{n+1}$. By $<_s$ condition c), $s_k <_s c_k$ for any node T_k in p. Thus, $s_i <_e c_i <_e s_k <_e c_j$.
    - $e = T_k \xrightarrow{rw} T_j$ is the last edge in p. By $<_s$ condition b), $s_k <_e c_j$. Any other edge $T_n \longrightarrow T_{n+1}$ in p is a dependency edge and, by $<_s$ condition a), $c_n <_e s_{n+1}$. By $<_s$ condition c), $s_k <_s c_k$ for any node T_k in p. Thus, $s_i <_e c_i <_e s_k <_e c_j$.

- $e = T_k \xrightarrow{rw} T_{k+1}$ is not the first, neither the last edge in p. Then, $T_k \neq T_i$ and $T_{k+1} \neq T_j$. Any other edge $T_n \longrightarrow T_{n+1}$ in p is a dependency edge and, by $<_s$ condition a), $c_n <_e s_{n+1}$. By $<_s$ condition c), $s_k <_s c_k$ for any node T_k in p. Thus, $c_i <_e s_k$ and $c_{k+1} <_e s_j$. Then $s_i <_e c_i <_e s_k <_e c_{k+1} <_e s_j <_e c_j$.

- **Induction hypothesis** $(n < l)$: the lemma holds if p has $n < l$ anti-dependency edges.
- **Induction step** $(n = l)$: assume now a path p starts from T_i, ends in T_j, has l anti-dependency edges and fulfils b) and c). Then $s_i <_s c_j$? Again, there are several possibilities:

  - p starts with an anti-dependency edge $e = T_i \xrightarrow{rw} T_k$. By $<_s$ condition b), $s_i <_e c_k$. Since there aren't two consecutive anti-dependency edges, the edge $T_k \longrightarrow T_{k+1}$ next to e is a dependency and, by $<_s$ condition a), $c_k <_e s_{k+1}$. The subpath $p' = T_{k+1} \longrightarrow ... \longrightarrow T_i$ has $l-1$ anti-dependency edges and, by the induction hypothesis, $s_{k+1} <_e c_j$. Thus, $s_i <_e c_k <_e s_{k+1} <_e c_j$.
  - p ends with an anti-dependency edge $e = T_k \xrightarrow{rw} T_j$. By $<_s$ condition b), $s_k <_e c_j$. Since there aren't two consecutive anti-dependency edges, the edge $T_{k-1} \longrightarrow T_k$ previous to e is a dependency and, by $<_s$ condition a), $c_k <_e s_{k+1}$. The subpath $p' = T_i \longrightarrow ... \longrightarrow T_{k-1}$ has $l-1$ anti-dependency edges and, by the induction hypothesis, $s_i <_e c_{k-1}$. Thus, $s_i <_e c_{k-1} <_e s_k <_e c_j$.
  - p does not start, neither ends with an anti-dependency edge. Imagine $e = T_k \xrightarrow{rw} T_{k+1}$ is one of the anti-dependency edges in p. Thus, by $<_s$ condition b), $s_k <_e c_{k+1}$. The previous and the next edges $T_{k-1} \longrightarrow T_k$ and $T_{k+1} \longrightarrow T_{k+2}$ are dependency edges because there aren't two consecutive anti-dependency edges. Thus, by $<_s$ condition a), $c_{k-1} <_e s_k$ and $c_{k+1} <_e s_{k+2}$. The paths from T_i to T_{k-1} and the path from T_{k+2} to T_j will have $l-1$ or less anti-dependency edges and, by the induction hypothesis, $s_i <_e c_{k-1}$ and $s_{k+2} <_e c_j$. Thus, $s_i <_e c_{k-1} <_e s_k <_e c_{k+1} <_e s_{k+2} <_e c_j$.

□

Lemma 5 (Start-dependency edges generated by SI-derived order $<_s$)**.** *Given a PL-SI' history H and a SI-derived order $<_s$, $c_i <_s s_j$ iff*

a) A path p in $DSG(H)$ connects T_i and T_j and

b) p is a directed path from T_i to T_j and

c) p does not contain two consecutive anti-dependency edges and

d) p starts and ends with dependency edges.

Proof. a) **If $c_i <_s s_j$ then a), b), c) and d)**: if $c_i <_s s_j$ then, by $<_s$ condition c), $s_i <_s c_j$ and Lemma 4 proves a), b), and c). Thus, there is a directed path p from T_i to T_j without two consecutive anti-dependency edges. We prove d) by induction over the length n of p:

- *Base case* $(n = 1)$: p is composed by a single directed edge e from T_i to T_j. Since $c_i <_s s_j$, by $<_s$ condition a) this edge must be a dependency edge.
- *Induction hypothesis* $(n < l)$: if p length $n < l$ and $c_i <_s s_j$, d) is also fulfilled.
- *Induction step* $(n = l)$: assume T_k and T_l are, either, the immediate nodes after T_i and before T_j in p. Thus, there is a directed path $p' = T_k \longrightarrow ... \longrightarrow T_l$ with length $n' < l$. If p' starts and ends with dependency edges, $c_k <_s s_l$ by the induction hypothesis. Otherwise, $s_k <_s c_l$ by Lemma 4. Since $c_i <_s s_j$, at least $c_i <_s s_k$ and $c_l <_s s_j$. Thus, by $<_s$ condition a), $T_i \longrightarrow T_k$ and $T_l \longrightarrow T_j$ must be dependency-edges and, thus, p starts and ends with dependency edges.

b) **If a), b), c) and d) then** $c_i <_s s_j$: thus a directed path p from T_i to T_j exists with an initial and final dependency edges and without two consecutive anti-dependency edges. By Lemma 4, $s_i <_s c_j$. We prove that also $c_i <_s s_j$ by induction over the length n of p:

- *Base case* $(n = 1)$: in that case, by d) p is composed by a single dependency edge. Thus, $c_i <_s s_j$ by $<_s$ condition b).
- *Induction hypothesis* $(n < l)$: if p length $n < l$ and a), b), c) and d) conditions are ensured, $c_i <_s s_j$.
- *Induction step* $(n = l)$: a), b), c) and d) and p length is $n = l$. $c_i <_s s_j$? p is a directed path $T_i \longrightarrow ... \longrightarrow T_j$ without two consecutive anti-dependency edges. p also starts and ends with $T_i \overset{ww/wr}{\longrightarrow} T_k$ and $T_l \overset{ww/wr}{\longrightarrow} T_j$ dependency edges. Thus, $T_k \longrightarrow ... \longrightarrow T_l$ is a path p' of length $n' < l$ which ensures a), b) and c) and, by Lemma 4 $s_k <_s c_l$. From $<_s$ condition b) we also get that $c_i <_s s_k$ and $c_l <_s s_j$. Thus, $c_i <_s s_k <_s c_l <_s s_j$.

□

Lemma 6 ($<_s$ conditions a), b) and c) are not contradictory)**.** *Conditions a), b) and c) applied to a PL-SI' history H will never produce a contradiction like $s_i <_e c_j$ and $c_j <_e s_i$.*

Proof. By absurd reduction, given two transactions $T_i, T_j \in H$, if $s_i <_e c_j$ and $c_j <_e s_i$ then, by Lemmas 4 and 5, there is a path $p1 = Ti \longrightarrow ... \longrightarrow T_j$ in $DSG(H)$ and another path $p2 = T\,j \longrightarrow ... \longrightarrow T_i$ such that $p1$ and $p2$ do not have two consecutive anti-dependency edges and $p2$ start and end with dependency edges. Thus, $p1$ and $p2$ form a cycle in $DSG(H)$ without two consecutive anti-dependency edges. However, that contradicts the initial assumption because that sort of cycles are explicitly forbidden by PL-SI'. □

References

[1] A. Adya. *Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions*. PhD thesis, Massachusetts Institute of Technology, March 1999.

[2] Atul Adya, Barbara Liskov, and Patrick E. O'Neil. Generalized isolation level definitions. In *16th Intnl. Conf. on Data Engin. (ICDE)*, pages 67–78, San Diego, California, USA, March 2000. IEEE-CS Press.

[3] Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, and Patrick E. O'Neil. A critique of ANSI SQL isolation levels. In *Intnl. Conf. on Management of Data (SIGMOD)*, pages 1–10, San Jose, California, USA, May 1995. ACM Press.

[4] Josep M. Bernabé-Gisbert and Francesc D. Muñoz-Escoí. Extending mixed serialisation graphs to replicated environments. In *3rd Intnl. Conf. on Availab., Reliab. and Secur. (ARES)*, pages 369–375, Barcelona, Spain, March 2008. IEEE-CS Press.

[5] Josep M. Bernabé-Gisbert, Raúl Salinas-Monteagudo, Luis Irún-Briz, and Francesc D. Muñoz-Escoí. Managing multiple isolation levels in middleware database replication protocols. *Lect. Notes Comput. Sc.*, 4330:511–523, December 2006.

[6] Arthur J. Bernstein, Philip M. Lewis, and Shiyong Lu. Semantic conditions for correctness at different isolation levels. In *16th Intnl. Conf. Data Eng. (ICDE)*, pages 57–66, San Diego, California, USA, February 2000. IEEE-CS Press.

[7] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Publishing Company, Reading, Massachusetts, USA, 1987.

[8] S. Elnikety, W. Zwaenepoel, and F. Pedone. Database replication using generalized snapshot isolation. In *24th Intnl. Symp. on Reliable Distrib. Syst. (SRDS)*, pages 73–84, Orlando, Florida, USA, October 2005. IEEE-CS Press.

[9] Alan Fekete. Allocating isolation levels to transactions. In *24th Symp. Princ. Database Syst. (PODS)*, pages 206–215, Baltimore, Maryland, USA, June 2005. ACM Press.

[10] Alan Fekete, Dimitrios Liarokapis, Elizabeth J. O'Neil, Patrick E. O'Neil, and Dennis Shasha. Making snapshot isolation serializable. *ACM Trans. Database Syst.*, 30(2):492–528, 2005.

[11] Alan Fekete, Elizabeth J. O'Neil, and Patrick E. O'Neil. A read-only transaction anomaly under snapshot isolation. *SIGMOD Record*, 33(3):12–14, 2004.

[12] INCITS. *ANSI/INCITS 135-1992 (R1998).- Information Systems - Database Language - SQL*. International Committee for Information Technology Standards, 1101 K Street NW, Suite 610, Washington, DC 20005, USA, January 1992.

[13] B. Kemme and G. Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Trans. Database Syst.*, 25(3):333–379, September 2000.

[14] Y. Lin, B. Kemme, M. Patiño-Martínez, and R. Jiménez-Peris. Middleware based data replication providing snapshot isolation. In *Intnl. Conf. on Management of Data (SIGMOD)*, pages 419–430, Baltimore, Maryland, USA, June 2005. ACM Press.

[15] Yi Lin, Bettina Kemme, Ricardo Jiménez-Peris, Marta Patiño-Martínez, and José Enrique Armendáriz-Iñigo. Snapshot isolation and integrity constraints in replicated databases. *ACM Trans. Database Syst.*, 34(2), 2009.

[16] David B. Lomet, Roger S. Barga, Mohamed F. Mokbel, German Shegalov, Rui Wang, and Yunyue Zhu. Transaction time support inside a database engine. In *22nd Intnl. Conf. on Data Engin. (ICDE)*, page 35, Atlanta, GA, USA, April 2006. IEEE-CS Press.

[17] Diana Lorentz and Mary Beth Roeser. *Oracle Database SQL Language Reference 11g Release 2 (11.2)*. Oracle Corporation, 500 Oracle Parkway, Redwood Shores, CA 94065, USA, September 2011. Available at: http://www.oracle.com/pls/db112/to_pdf?pathname=server.112/e25513.pdf.

[18] Microsoft Development Network. *Microsoft SQL Server Books Online.- Transact-SQL Reference.- SET TRANSACTION ISOLATION LEVEL*. Microsoft Corporation, One Microsoft Way, Redmond, Washington, USA, March 2012. Available at: http://msdn.microsoft.com/library/ms173763.aspx.

[19] PostgreSQL G.D.G. *PostgreSQL 9.1.3 Documentation.- Chapter 13: Concurrency Control*. The PostgreSQL Global Development Group, March 2012. Available at: http://www.postgresql.org/docs/9.1/static/transaction-iso.html.

[20] Irving L. Traiger, Jim Gray, Cesare A. Galtieri, and Bruce G. Lindsay. Transactions and consistency in distributed database systems. *ACM Trans. Database Syst.*, 7(3):323–342, 1982.