An Architecture for Large Multidatabase Systems*

Sharad Mehrotra[1]
Henry F. Korth[2]
Avi Silberschatz[3†]

[1]Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712-1188 USA

[2]Matsushita Information Technology Laboratory
2 Research Way
Princeton, NJ 08540

[3]AT&T Bell Laboratories
600 Mountain Ave.
Murray Hill, NJ 07974

Abstract

Over the past decade, substantial research has been done towards developing transaction management algorithms for multidatabase systems. Most of these research efforts have concentrated on the problems that arise due to the *heterogeneity* and the *autonomy* of the various local databases that are integrated into a multidatabase environment. One issue that has been relatively ignored is that of the architecture of multidatabase systems. We believe that a large multidatabase system spanning multiple organizations that are distributed over various geographically distant locations will not be developed as a single monolithic system. Rather, it will be developed hierarchically. As a result, the transaction management algorithms followed by a multidatabase system must be *composable* in such a way that it is feasible to incorporate individual multidatabase systems as elements in a larger multidatabase system. In this paper, we present a hierarchical architecture for a multidatabase environment, and develop a methodology for the design of composable transaction management algorithms suited for this architecture.

1 Introduction

A multidatabase system (MDBS) is a facility, developed on top of pre-existing local database management systems (DBMSs), that provides users of a DBMS access and update privileges to data located in other heterogeneous data sources. The following two characteristics of the MDBS environments make the task of designing transaction management algorithms difficult:

*Work partially supported by NSF grants IRI-8805215, IRI-9003341 and IRI-9106450, and by a grant from the IBM corporation.

†On leave from The University of Texas at Austin.

- **Heterogeneity.** Each local DBMS may follow different concurrency control protocols and recovery algorithms.
- **Autonomy.** The participation of the local DBMS in an MDBS must not result in a loss of control by the local DBMS over its data and its local transactions.

Over the past decade, substantial research has been done to identify mechanisms for effectively dealing with the problems that arise due to the heterogeneity and the autonomy of the local systems (e.g., [BST90, WV90, MRB+92b, Pu88, ED90, MRB+92a, BS88]). This research has resulted in transaction management algorithms that ensure correctness without sacrificing the autonomy of the individual systems. Most of the proposed approaches have, however, considered an MDBS as a single monolithic system which executes on top of the existing local DBMSs and controls the execution and commitment of the *global transactions* (transactions that execute at multiple local DBMSs) in such a way that consistency of the individual system is not jeopardized.

One issue that has been given relatively little consideration is that of the architecture of MDBSs. We believe that a large MDBS, that spans multiple organizations distributed over various geographically distant locations, will not be developed as a single monolithic system. Instead it will be developed hierarchically. To illustrate this, let us consider a typical MDBS environment in which users wish to execute transactions that span database systems belonging to multiple branches of an organization. Additionally, users also wish to execute transactions that span different autonomous organizations. One solution to providing such a service is to develop a single monolithic MDBS system which integrates all the branches of all the organizations. However, depending upon the nature of transactions that execute within an organization, the computing resources available, and the reliability of the network, different organizations may prefer different MDBS transaction management algorithms for processing transactions local within the organization. For example, if a high degree of concurrency is critical for good performance in a certain organization, that organization may prefer a centralized MDBS transaction management algorithm for processing transactions local within the organization. On the other hand, if databases belonging to various branches of another organization are geographically distant and the network is not reliable, the organization may prefer a fully decentralized MDBS transaction management algorithm for processing transactions that execute within its different branches. Thus, it would be preferable to develop the MDBS as a hierarchical system– each organization (or a set of organizations) has its own MDBS to control the execution of transactions within the organization. Furthermore, an inter-organization MDBS controls the execution of transactions that access data belonging to branches of different organizations. Note that using a single monolithic MDBS system, whether distributed or centralized, will adversely impact the performance of transactions that execute within an organization. In contrast, in a hierarchical MDBS, each organization can use a specialized transaction management algorithm suited for their environment.

The above scenario illustrates why it would be desirable for the MDBS architecture to be hierarchical. However, if the architecture of the MDBS is hierarchical, then the transaction management algorithms followed by individual MDBSs need to be *composable* in such a way that it is feasible to incorporate individual MDBSs as elements in a larger MDBS. In this paper, we present a hierarchical architecture for multidatabase systems. We adopt serializability as the correctness criterion and develop a methodology for the design of composable transaction management algorithms that ensures global serializability in hierarchical MDBSs.

The rest of the paper is organized as follows. In Section 2, we formally define our MDBS architecture. In Section 3, we review how the problem of heterogeneity is overcome in MDBSs. In Section 4, we develop a methodology for designing transaction management algorithms suited for our MDBS architecture. In Section 5, we identify restrictions on the architecture such that concurrency control schemes that follow our methodology result in global serializability. Finally, in Section 6, we offer concluding remarks and present directions for future work.

2 MDBS Architecture

An MDBS is an integrated collection of pre-existing local databases: DBMS_1, DBMS_2, ..., DBMS_m, that permits users to execute transactions that access multiple local DBMSs. Each local DBMS may itself be either a centralized or a distributed database system. Each DBMS_i contains a set of data items that are denoted by DB_i. To describe the architecture of the MDBS, we associate with the MDBS environment a set of *domains* denoted by Δ with an ordering relation $\sqsubset$. A domain $D \in \Delta$ is either

- a set of data items in DB_i, for some $i = 1, 2, \ldots, m$, or
- a union of the set of data items in other domains $D_1, D_2, \ldots, D_n$, denoted by $\bigcup\{D_1, D_2, \ldots, D_n\}$, where $D_i \in \Delta$, $i = 1, 2, \ldots, n$,

The ordering relation $\sqsubset$, referred to as the *domain ordering* relation, is such that $D_i \sqsubset D_j$ iff $D_i \subset D_j$. We use $D_i \sqsubseteq D_j$ to denote that either $D_i \sqsubset D_j$ or $D_i = D_j$. Let D_i and D_j be domains in Δ. We refer to D_i as the child of D_j, denoted by $child(D_i, D_j)$, if $D_i \sqsubset D_j$ and for all $D_k \in \Delta$, either $D_i \not\sqsubset D_k$ or $D_k \not\sqsubset D_j$. Further, we refer to D_j as a parent of D_i, denoted by $parent(D_j, D_i)$, if $child(D_i, D_j)$. We denote the set of domains $\{D| \text{ for all } D_k \in \Delta, D \not\sqsubset D_k\}$ by the set TOP.

A transaction $T_i = (O_{T_i}, \prec_{T_i})$, where O_{T_i} is the set of operations and $\prec_{T_i}$ is a partial order over operations in O_{T_i}. We assume that a transaction T_i that execute at a local DBMS (or a set of local DBMSs) consist of a set of **read** (denoted by r_i) and **write** (denoted by w_i) operations. Further, each transaction T_i has **begin** (denoted by b_i) and **commit** (denoted by c_i) operations. A transaction that executes at multiple DBMSs may have multiple begin and commit operations[1], one

[1]In contrast, the r_i and w_i operations of the transaction on each data item are unique. Since, in this paper, we do not consider the problem of replica control, we consider different copies of the same data item as independent data items with an equality constraint between them.

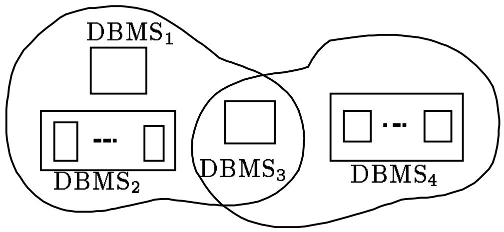

Figure 1(a): An Example MDBS Environment

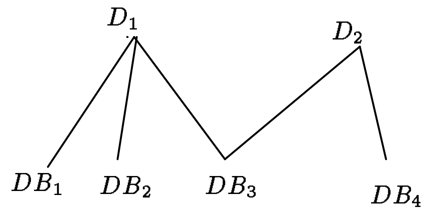

Figure 1(b): Domain Ordering for Figure 1(a)

for each DBMS at which it executes. We denote by b_{ik} and c_{ik}, the begin and commit operations of a transaction T_i in DBMS$_k$ respectively.

A transaction T_i is said to execute in a domain $D \in \Delta$, if there exists a DB_j, $DB_j \sqsubseteq D$, such that T_i accesses data items in DB_j. A transaction T_i may execute in multiple domains subject to the following restriction. If T_i accesses data items in $DB_1, DB_2, \ldots, DB_k$, then there must exist a domain $D \in \Delta$ such that $DB_j \sqsubseteq D$, $j = 1, 2, \ldots, k$. Such a domain D is denoted by $Dom(T_i)$. Thus, if T_i accesses data item in DB_j, then $DB_j \sqsubseteq Dom(T_i)$. A transaction T_i is said to be *global* with respect to a domain $D \in \Delta$, denoted by $global(T_i, D)$, if T_i executes in D and there exists a domain D', $D' \not\sqsubseteq D$ and $D \not\sqsubseteq D'$ such that T_i executes in D'. A transaction T_i is local with respect to a domain D, denoted by $local(T_i, D)$, if T_i executes in D and $\neg global(T_i, D)$. We illustrate the above defined notations by the following example.

Example 1: Consider an MDBS environment consisting of four local DBMSs – DBMS$_1$, DBMS$_2$, DBMS$_3$, DBMS$_4$ (illustrated in Figure 1(a)). DBMS$_1$ and DBMS$_3$ are centralized database systems, while DBMS$_2$ and DBMS$_4$ are distributed database systems. The set of domains, $\Delta = \{DB_1, DB_2, DB_3, DB_4, D_1, D_2\}$, where Domain $D_1 = \bigcup\{DB_1, DB_2, DB_3\}$ and domain $D_2 = \bigcup\{DB_3, DB_4\}$. The domain ordering relation for the MDBS environment depicted in Figure 1(a) is illustrated in Figure 1(b).

Consider a transaction T_1 that accesses data items in domains DB_1 and DB_2. Thus, $Dom(T_1) = D_1$, $global(T_1, DB_1)$, $global(T_1, DB_2)$, and $local(T_1, D_1)$. Consider another transaction T_2 that accesses data in domains DB_3 and DB_4; thus, $global(T_2, D_1)$, $global(T_2, D_2)$ and $Dom(T_2) = D_2$. Finally, consider a transaction T_3 that wishes to access data in DB_1 and DB_4. T_3 will not be permitted to execute since there does not exist any domain $D \in \Delta$ such that $DB_1 \sqsubset D$ as well as $DB_4 \sqsubset D$. However, if there was a domain $D_3 = \bigcup\{D_1, D_2\}$, then the transaction T_3 would be permitted and $Dom(T_3) = D_3$. $\Box$

Let $S = (\tau_S, \prec_S)$ be a schedule, where τ_S is a set of transactions and $\prec_S$ is a partial order over the operations belonging to transactions in τ_S. The partial order $\prec_S$ satisfies the property that

$\prec_{T_i} \subseteq \prec_S$, for each $T_i \in \tau_S$. Let d be a set of data items. S^d denotes the projection of S onto data items in d. Formally, schedule S^d is a *restriction*[2] of the schedule S over the set of data items in d. For notational brevity, we denote the projection of S over the set of data items in DB_k; that is, S^{DB_k}, by S_k.

In a schedule $S = (\tau_S, \prec_S)$, transactions $T_i, T_j \in \tau_S$ are said to *conflict* in S, denoted by $T_i \leadsto_S T_j$, if there exists operations o_i in T_i and o_j in T_j such that o_i and o_j *conflict* in S and $o_i \prec_S o_j$. Operations o_i and o_j are said to conflict if they access the same data item and at least one of them is a write operation. We denote the transitive closure of the conflict relation $\leadsto$ among transactions by the relation $\overset{*}{\leadsto}$.

With each domain D_i a *domain manager* $DM(D_i)$ is associated. The domain manager for a domain D_i, along with the domain managers of each domain D_j, $D_j \sqsubset D_i$, controls the concurrent execution of transactions that execute in D_i in such a way that the consistency of data within a domain is preserved. Let D be a domain such that $DB_j \sqsubset D$, $j = 1, 2, \ldots, k$. The domain managers of the domains $D' \sqsubseteq D$, in our architecture, constitute the MDBS software for an MDBS that integrates DBMS_1, DBMS_2, $\ldots$, DBMS_k. Note that if there exists a domain $D \in \Delta$ such that for each DB_k, $k = 1, 2, \ldots, m$, $parent(D, DB_k)$, then our MDBS architecture reduces to a single monolithic system. In this case, the existing solutions for transaction management developed for such systems in [MRB+92a, ED90, BS88, BST90] can be used by the domain manager for D to control the concurrent execution of the transactions. Similarly, if we were to restrict Δ such that

$$\text{for all domains } D_i, D_j \in \Delta, \text{ if } child(D_i, D_j), \text{ then for all } D_k \neq D_j, \neg child(D_i, D_k),$$

then our MDBS architecture reduces to the *superdatabase* architecture for MDBSs that was developed in [Pu88] and the algorithms for concurrency control developed there can then be used. However, our proposed solutions differ widely from the concurrency control algorithms suggested in [Pu88].

3 Background

Before discussing how concurrency control for ensuring global serializability can be done in hierarchical MDBSs, we first review how global serializability can be ensured if an MDBS were to be developed as a single monolithic system. Crucial to the development of the concurrency control protocols is the notion of serialization functions introduced in [MRB+92a] which is similar to the notion of *o-element* developed in [Pu88].

Let $S = (\tau_S, \prec_S)$ be a serializable schedule. Let $\tau' \subseteq \tau_S$. A serialization function of a transaction $T_i \in \tau'$ in a schedule S with respect to the set of transactions τ', denoted by $ser_{S,\tau'}(T_i)$ is a function that maps $T_i \in \tau'$ to some operation in T_i such that the following holds:

$$\text{For all } T_i, T_j \in \tau', \text{ if } T_i \overset{*}{\leadsto}_S T_j, \text{ then } ser_{S,\tau'}(T_i) \prec_S ser_{S,\tau'}(T_j)$$

[2] A set P_1 with a partial order $\prec_{P_1}$ on its elements is a *restriction* of a set P_2 with a partial order $\prec_{P_2}$ on its elements if $P_1 \subseteq P_2$, and for all $e_1, e_2 \in P_1$, $e_1 \prec_{P_1} e_2$ if and only if $e_1 \prec_{P_2} e_2$.

In the remainder of the paper, we will denote the function $ser_{S,\tau'}$ by ser_S. The set of transactions τ' will be clear from the context. For numerous concurrency control protocols that generate serializable schedules, it is possible to associate a serialization function with transactions T in the schedule S such that the above property is satisfied.

For example, if the *timestamp ordering* (TO) concurrency control protocol is used to ensure serializability of S and the scheduler assigns timestamps to transactions when they begin execution, then the function that maps every transaction $T_i \in \tau_S$ to T_i's begin operation is a serialization function for transaction T_i in S with respect to the set of transactions τ_S.

For a schedule S, there may be multiple serialization functions. For example, if S is generated by a the *two-phase locking* (2PL) protocol, then a possible serialization function for transactions in S maps every transaction $T_i \in \tau_S$ to the operation that results in T_i obtaining its last lock. Alternatively, the function that maps every transaction $T_i \in \tau_S$ to the operation that results in T_i releasing its first lock is also a serialization function for T_i in S[3].

It is possible that for transactions in a schedule generated by certain concurrency control protocols, no serialization function may exist. Consider, for example, a schedule generated by *serialization-graph testing* (SGT) scheduler. In this case, it may not be possible to associate a serialization function with transactions. However, in such schedules, serialization functions can be introduced by forcing direct conflicts between transactions [GRS91]. Let $\tau' \subseteq \tau$ be some set of transactions in a schedule S. If each transaction in τ' executed a conflicting operation (say a write operation on data item *ticket*), in S, then the functions that maps a transaction $T_i \in \tau'$ to its write operation on *ticket* is the serialization function for the transactions in S with respect to the set of transactions τ'.

Associating serialization functions with transactions enables us to overcome the problems due to heterogeneity of local DBMSs in designing concurrency control protocols for ensuring global serializability in an MDBS environment. To see this, let us consider a collection of local DBMSs, DBMS_1, DBMS_2, ..., DBMS_m, which are to be integrated into an MDBS. Each local DBMS, DBMS_k follows some concurrency control protocol to ensure serializability of its local schedule S_k. Let $\tau_k = \{T_i \mid global(T_i, DB_k)\}$, be the set of global transactions (transactions that access data residing in other databases besides DB_k) in S_k, $k = 1, 2, \ldots, m$. We assume that a serialization function can be associated with global subtransactions in each schedule S_k with respect to the transactions in τ_k (introduced, if necessary, using external means by forcing direct conflicts between transactions in τ_k).

Let T_i be a global transaction. We denote the projection of T_i to its serialization function values over each of the local schedules as a transaction $\widehat{T}_i$. Thus, $\widehat{T}_i$ is a restriction of T_i consisting of all the operations in the set $\{ser_{S_k}(T_i) \mid T_i \in \tau_k\}$. For the global schedule S, we denote a restriction of S consisting of the set of operations belonging to transactions $\widehat{T}_i$ by $\widehat{S}$. Thus, $\widehat{S} = (\tau_{\widehat{S}}, \prec_{\widehat{S}})$, where $\tau_{\widehat{S}} = \{\widehat{T}_i \mid T_i \in \tau_k,$ for some $k = 1, 2, \ldots, m\}$. Furthermore, $\prec_{\widehat{S}} \subseteq \prec_S$. In the schedule $\widehat{S}$, we define

[3]Actually, any function that maps a transaction $T_i \in \tau_S$ to one of its operations that executes between the time T_i obtains its last lock and the time it releases its first lock is a serialization function for T_i in S.

operations $ser_{S_k}(T_i)$ and $ser_{S_l}(T_j)$, $T_i \neq T_j$, to conflict iff $k = l$. Let us illustrate the above notation with an example.

Example 2: Consider an MDBS environment consisting of two local databases. DBMS_1 contains data items a and b, while DBMS_2 contains data item c. Suppose that DBMS_1 follows the TO scheme in which a timestamp is assigned to a transaction when it begins execution, and DBMS_2 follows the strict 2PL protocol [BHG87] for ensuring serializability of its local schedules. Consider the following global transactions T_1 and T_2 that execute.

$$T_1 : \; b_{11} \;\; w_1(a) \;\; b_{12} \;\; w_1(c) \;\; c_{11} \;\; c_{12}$$

$$T_2 : \; b_{21} \;\; r_2(b) \;\; b_{22} \;\; r_2(c) \;\; c_{21} \;\; c_{22}$$

Let T_3 be a transaction local to DBMS_1.

$$T_3 : \; b_3 \;\; r_3(a) \;\; w_3(b) \;\; c_3$$

Consider the global schedule S resulting from the concurrent execution of transaction T_1, T_2 and T_3 such that the local schedules at DBMS_1 and DBMS_2 are as follows.

$$S_1 : \; b_{11} \;\; b_3 \;\; w_1(a) \;\; b_{21} \;\; r_3(a) \;\; w_3(b) \;\; c_3 \;\; r_2(b) \;\; c_{11} \;\; c_{21}$$
$$S_2 : \; b_{22} \;\; b_{12} \;\; w_1(c) \;\; c_{12} \;\; r_2(c) \;\; c_{22}$$

Let $\tau_1 = \{T_1, T_2\}$ and $\tau_2 = \{T_1, T_2\}$ be the set of global transactions executing on databases DBMS_1 and DBMS_2 respectively. Let ser_{S_1} be the function that maps every transaction in τ_1 to its begin operation. Also, let ser_{S_2} be the function that maps every transaction in τ_2 to its commit operation. Thus, $ser_{S_1}(T_1) = b_{11}$, $ser_{S_1}(T_2) = b_{21}$, $ser_{S_2}(T_1) = c_{12}$ and $ser_{S_2}(T_2) = c_{22}$. As a result, transactions $\widehat{T}_1$, $\widehat{T}_2$ are as follows.

$$\widehat{T}_1 : \; b_{11} \;\; c_{12}$$

$$\widehat{T}_2 : \; b_{21} \;\; c_{22}$$

Schedule $\widehat{S}$ is as follows.

$$\widehat{S} : \; b_{11} \;\; b_{21} \;\; c_{12} \;\; c_{22}$$

In $\widehat{S}$, operations b_{11} and b_{21} conflict, whereas operations b_{11} and c_{22} do not conflict. Note that operations b_{11} and b_{21} do not conflict in S. $\Box$

Theorem 1: [MRB+92a] Consider an MDBS consisting of DBMS_1, DBMS_2, ..., DBMS_m. Let S be a global schedule. S is serializable, if the schedule $\widehat{S}$ is serializable. $\Box$

In Example 2, note that $\widehat{S}$ is serializable (the serialization order being $\widehat{T}_1$ before $\widehat{T}_2$). As a result, global schedule S is serializable. Theorem 1 reduces the problem of ensuring global serializability to the problem of ensuring the serializability of the schedule $\widehat{S}$. Note that each operation in the schedule $\widehat{S}$ belongs to only global transactions. Thus, the MDBS can guarantee global serializability by ensuring that the order in which these operations execute the resulting schedule $\widehat{S}$ is serializable. Note that the schedule $\widehat{S}$ is distributed over the local DBMSs. Thus, the MDBS transaction manager can employ any distributed or centralized concurrency control protocol for ensuring serializability of $\widehat{S}$. For example, the scheme suggested in [GRS91] uses a SGT certifier to ensure serializability of $\widehat{S}$. On the other hand, the scheme suggested in [ED90] uses a TO scheme to ensure serializability of $\widehat{S}$. Further, the scheme suggested in [BGR92] uses a distributed TO protocol to ensure serializability of $\widehat{S}$. In [MRB+92a] we suggested various conservative schemes to ensure serializability of $\widehat{S}$.

4 Concurrency Control in Hierarchical MDBSs

In a hierarchical MDBS, the domain manager for a domain D, along with the domain manager for each domain D', $D' \sqsubseteq D$, controls the concurrent execution of the transactions that execute in D. In order to ensure global serializability, domain manager $DM(D)$ for each domain D must ensure that the concurrent execution of transactions in D results in a serializable schedule. In this section, we propose a mechanism that $DM(D)$ can use in order to ensure the serializability of schedules resulting from the concurrent execution of transactions within D. Crucial to our development is the appropriate extension of the notion of serialization functions to the domain. Let D be any arbitrary domain in Δ. An extended serialization function is a function sf that maps a given transaction T_i, and a domain D, to some operation belonging to T_i that executes in D such that the following holds:

For all T_i, T_j, if $global(T_i, D)$, $global(T_j, D)$, and $T_i \overset{*}{\leadsto}_{S^D} T_j$, then $sf(T_i, D) \prec_{S^D} sf(T_j, D)$

We refer to $sf(T_i, D)$ as a serialization function of transaction T_i with respect to the domain D. To see how such a serialization function will aid us in ensuring serializability within a domain, consider a domain $D \neq DB_k$, $k = 1, 2, \ldots, m$. Let us assume that the above defined serialization function exists for transactions in every child domain of D; that is, for every D_k, where $child(D_k, D)$. For a given transaction T_i that executes in D, we denote the projection of T_i to its serialization function values over each of the child domains of D as a transaction $\widehat{T}_i^D$. Formally, $\widehat{T}_i^D$ is defined as follows. Let T_i be a transaction and D be a domain such that $global(T_i, D_k)$ for some D_k, where $child(D_k, D)$. $\widehat{T}_i^D$ is a restriction of T_i consisting of all the operations in the set $\{sf(T_i, D_k) \mid T_i \text{ executes in } D_k, \text{ and } child(D_k, D)\}$. Further, for the global schedule S, we define a schedule $\widehat{S}^D$ to be the restriction of S consisting of the set of operations belonging to transactions $\widehat{T}_i^D$. Thus, $\widehat{S}^D = (\tau_{\widehat{S}^D}, \prec_{\widehat{S}^D})$, where $\tau_{\widehat{S}^D} = \{\widehat{T}_i^D \mid global(T_i, D_k) \text{ for some } D_k, \text{ where } child(D_k, D)\}$, and for all operations o_q, o_r in $\widehat{S}^D$, $o_q \prec_{\widehat{S}^D} o_r$, iff $o_q \prec_S o_r$. In the schedule $\widehat{S}^D$, we define operations $sf(T_i, D_k)$ and $sf(T_j, D_l)$, $T_i \neq T_j$, to conflict iff $k = l$.

Lemma 1: Consider an MDBS environment with the set Δ of domains. Let S be a global schedule and D be an arbitrary domain in Δ. Schedule S^D is serializable, if each of the following three conditions hold:

- For each domain D_k such that $child(D_k, D)$, S^{D_k} is serializable.
- For each domain D_k, such that $child(D_k, D)$, there exists a serialization function sf such that the following holds:

  For all transactions T_i, T_j, if $global(T_i, D_k)$, $global(T_j, D_k)$, and $T_i \overset{*}{\rightsquigarrow}_{S^{D_k}} T_j$, then $sf(T_i, D_k) \prec_S sf(T_j, D_k)$.
- Schedule $\widehat{S}^D$ is serializable. $\square$

Lemma 1 demonstrates that if an appropriate serialization function is associated with child domains of a domain D, then serializability of the projection of the schedule S to domain D can be ensured. We, therefore, need to associate an appropriate serialization function with each domain $D \in \Delta$. Note that for a domain $D = DB_k$, the function sf is simply ser_{S_k}. We now define the function sf for an arbitrary domain $D \in \Delta$, which is done recursively over the domain ordering relation.

Definition 1: Let D be a domain and T_i be a transaction such that $global(T_i, D)$. The serialization function for transaction T_i in domain D is defined as follows:

$$sf(T_i, D) = \begin{cases} ser_{S_k}(T_i), & \text{if for some } DB_k,\ \ D = DB_k. \\ ser_{\widehat{S}^D}(\widehat{T}_i^D), & \text{if for all } DB_k,\ \ D \neq DB_k \end{cases} \quad \square$$

We next show that the above defined function sf indeed meets our requirement of a serialization function for a domain D.

Lemma 2: Consider an MDBS environment with the set Δ of domains. Let S be a global schedule, T_i, T_j be transactions in S, and D be an arbitrary domain in Δ. If $global(T_i, D)$, $global(T_j, D)$ and $T_i \overset{*}{\rightsquigarrow}_{S^D} T_j$, then $sf(T_i, D) \prec_S sf(T_j, D)$. $\square$

Using Lemmas 1 and 2, we can show that if for each domain D', $D' \sqsubseteq D$, the schedule $\widehat{S}^{D'}$ is serializable, then the schedule S^D is serializable. This is stated in the following theorem.

Theorem 2: Consider an MDBS environment with the set Δ of domains. Let S be a global schedule and D be an arbitrary domain in Δ. Schedule S^D is serializable, if the following three conditions hold:

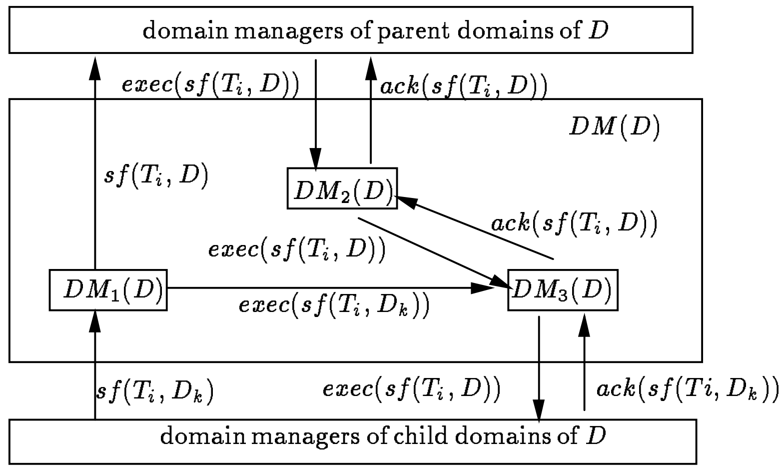

Figure 2: Components of a Domain Manager

- For each DB_k such that $DB_k \sqsubset D$, S_k is serializable and further there exists a function ser_{S_k} such that for all transactions T_i, T_j, if $global(T_i, DB_k)$, $global(T_j, DB_k)$, and $T_i \overset{*}{\leadsto}_{S_k} T_j$, then $ser_{S_k}(T_i) \prec_S ser_{S_k}(T_j)$.
- For all domains $D' \in \Delta$ such that $D' \sqsubset D$, $\widehat{S}^{D'}$ is serializable and further there exists a function $ser_{\widehat{S}^{D'}}$ such that for all transactions T_i, T_j, if $global(T_i, D')$, $global(T_j, D')$, and $\widehat{T}_i^{D'} \overset{*}{\leadsto}_{\widehat{S}^{D'}} \widehat{T}_j^{D'}$, then $ser_{\widehat{S}^{D'}}(\widehat{T}_i^{D'}) \prec_S ser_{\widehat{S}^{D'}}(\widehat{T}_j^{D'})$.
- $\widehat{S}^D$ is serializable. $\Box$

Theorem 2 states that in order to ensure the serializability of the schedule S^D, the domain manager of each domain $D' \sqsubseteq D$ needs to ensure that the schedule $\widehat{S}^{D'}$ is serializable. Let D be an arbitrary domain in Δ such that $D \notin TOP$ and $D \neq DB_k$, for all $k = 1, 2, \ldots, m$. We next consider a design of the domain manager for such a domain D (denoted by $DM(D)$) that ensures the serializability of the schedule $\widehat{S}^D$. Domain managers for domains D such that $D \in TOP$ or $D = DB_k$, for some $k = 1, 2, \ldots, m$, are a slight modifications of the basic design and are discussed later.

$DM(D)$ consists of three components – $DM_1(D)$, $DM_2(D)$ and $DM_3(D)$ (see Figure 2). Components $DM_1(D)$ and $DM_2(D)$ together are responsible for submitting the operations belonging to the transactions $\widehat{T}_i^D$ to the component $DM_3(D)$. Component $DM_3(D)$ schedules the operations belonging to transactions $\widehat{T}_i^D$ in such a fashion that the schedule $\widehat{S}^D$ is serializable.

- $DM_1(D)$**:** The component $DM_1(D)$ is responsible for forwarding the requests from the domain manager of the child domains of D to either the parent domains of D, or to the component $DM_3(D)$. $DM_1(D)$ receives operations $o = sf(T_i, D_k)$ from the domain manager of D_k, where $child(D_k, D)$. It uses the information about the concurrency control protocol followed by

$DM_3(D)$ to determine if the operation is the serialization function of T_i with respect to the domain D; that is, if $o = sf(T_i, D)$. If the transaction T_i is local to D (that is, $local(T_i, D)$), or if $o \neq sf(T_i, D)$, $DM_1(D)$ submits a request for the execution of the operation $sf(T_i, D_k)$ (denoted by $exec(sf(T_i, D_k))$) to $DM_3(D)$. Else, if T_i is global to D (that is, $global(T_i, D)$), and $o = sf(T_i, D)$, then it submits the operation to the domain managers of every domain D' such that $parent(D', D)$. Recall that a domain D, in our MDBS architecture, may have multiple domains D' such that $parent(D', D)$.

- $DM_2(D)$: The component $DM_2(D)$ is responsible for collecting requests for the execution of operations $o = sf(T_i, D)$ from the parent domains of D. $DM_2(D)$ receives requests for the execution of the operations $o = sf(T_i, D)$ (that is, $exec(sf(T_i, D))$ requests) from the domain managers of the domains D', where $parent(D', D)$. In case there are multiple domains D' such that $parent(D', D)$, $DM_2(D)$ waits until it receives requests $exec(sf(T_i, D))$ from each domain D', where $parent(D', D)$. On receipt of the request from *each* of the parent domains, it submits the operation for execution to the component $DM_3(D)$. On receipt of the acknowledgement for the successful execution of the operation $sf(T_i, D)$ (denoted by $ack(sf(T_i, D))$) from $DM_3(D)$, $DM_2(D)$, in turn, forwards the acknowledgement to the domain managers of each of the domains D', where $parent(D', D)$.

- $DM_3(D)$: The component $DM_3(D)$ is responsible for scheduling the operations of the transactions $\widehat{T}_i^D$ in such a fashion that the schedule $\widehat{S}^D$ is serializable. $DM_3(D)$ receives request for the execution of operations $o = sf(T_i, D_k)$, where $child(D_k, D)$ from $DM_1(D)$ (if either o belongs to a transaction T_i such that $local(T_i, D)$, or if $o \neq sf(T_i, D)$) and from the component $DM_2(D)$ (if $o = sf(T_i, D)$, and $global(T_i, D)$). $DM_3(D)$, in turn, submits the request for the execution of the operation $sf(T_i, D_k)$, to the domain manager of the domain D_k, where $child(D_k, D)$. Further, on receipt of the acknowledgement for the operation $o = sf(T_i, D_k)$ (that is, $ack(sf(T_i, D_k))$) from the domain manager of the domain D_k, in case the operation is also the serialization function of T_i with respect to D (that is, $sf(T_i, D)$), $DM_3(D)$ forwards the acknowledgement to the component $DM_2(D)$ which, as mentioned previously, acknowledges the execution of the operation to the domain managers of each of the parent domains of D. $DM_3(D)$ controls the submission order of the operations $sf(T_i, D_k)$ to the domain managers of the domains D_k, where $child(D_k, D)$, in such a fashion that the schedule $\widehat{S}^D$ is serializable.

The domain manager for the domain $D \in TOP$ differs from the above in that it does not contain the component $DM_2(D)$. Note that if $D \in TOP$, then there does not exist a domain D' such that $parent(D', D)$. Thus, the component $DM_1(D)$ of the domain manager for a domain $D \in TOP$, on receipt of the any operations $o = sf(T_i, D_k)$, where $child(D_k, D)$, submits a request for the execution of $sf(T_i, D_k)$ (that is, $exec(sf(T_i, D_k))$ to the component $DM_3(D)$ directly.

The domain manager for the domain $D = DB_k$, for some $k = 1, 2, \ldots, m$, differs from the design of the domain manager illustrated in Figure 2 in that it does not contain the component

$DM_3(D)$. In this case, the request for the execution of the operations $sf(T_i, D)$ by the components $DM_1(D)$ and $DM_2(D)$ are submitted directly to the local DBMS for execution. We assume that each local DBMS, on successful execution of the operation, acknowledges its execution to $DM(D)$. Furthermore, unlike the case for other domains, operations belonging to transactions that are local to D (that is, $local(T_i, D)$) are not controlled by the domain manager of D. Instead, they are submitted to the local DBMS directly for execution. In contrast, operations belonging to the transactions that are global to D, that is, $global(T_i, D)$, as in the case of other domains, are controlled by the domain manager. For each transaction T_i such that $global(T_i, D)$, the component $DM_1(D)$, based on the concurrency control protocol followed by the local DBMS to ensure serializability of the schedule S^D, determines whether the operation is the serialization function of T_i with respect to D. If the operation $o = sf(T_i, D)$, then, as before, $DM_1(D)$ forwards the operation to the domain managers of all domains D', where $parent(D', D)$. Else, it submits the operation to the local DBMS for execution. The local DBMS, on receipt of the operation from $DM(D)$ executes the operation. Further, on completion of the execution of the operation, it acknowledges its execution to the domain manager $DM(D)$. $DM_2(D)$, on receipt of the acknowledgement of the operation from the local DBMS, determines if the acknowledgement is for an operation $sf(T_i, D)$. If the acknowledgement if for the operation $sf(T_i, D)$, then as in the case of other domain managers, $DM_2(D)$ sends an acknowledgement to the domain managers of the parent domains of D.

In our design of the domain manager for a domain D, the operation $o = sf(T_i, D_k)$ does not execute in S until the component $DM_3(D)$ of the domain manager for domain D submits a request for the execution of the operation $sf(T_i, D_k)$; that is, $exec(sf(T_i, D_k))$ to the domain manager of domain D_k, where $child(D_k, D)$. Note that this is true since the component $DM_2(D_k)$ of the domain manager for the child domain D_k waits to receive a request for the execution of the operation $sf(T_i, D_k)$ from each parent domain of D_k. Furthermore, for each operation $sf(T_i, D_k)$, the component $DM_3(D)$ of the domain manager for the domain D receives the acknowledgement for the execution of $sf(T_i, D_k)$, where $child(D_k, D)$, sometime after the execution of $sf(T_i, D_k)$ in S. This is true since we assume that each DBMS_j acknowledges the execution of the operations belonging to the transactions that are global with respect to DB_j to the domain manager of $D = DB_j$, and the domain manager for each domain D, in turn, acknowledges the execution of the operation $sf(T_i, D)$, to the domain managers of each of its parent domains. Thus, the operation $sf(T_i, D_k)$ executes in S after $DM_3(D)$ submits $sf(T_i, D_k)$ for execution to the domain manager of D_k, and before $DM_3(D)$ receives the acknowledgement for the execution of $sf(T_i, D_k)$ from the domain manager of D_k. Hence, to ensure that the schedule $\widehat{S}^D$ is serializable, the component $DM_3(D)$ can use any concurrency control protocol that ensures serializability (e.g., 2PL, TO, SGT) to schedule the submission of the operations belonging to transactions $\widehat{T}_i^D$ to the domain managers of the child domains. Note that since the schedule $\widehat{S}^D$ is distributed over the domains $D_1, D_2, \ldots, D_k$, where $child(D_j, D)$, $j = 1, 2, \ldots, k$, $DM_3(D)$ can follow any distributed or centralized concurrency control

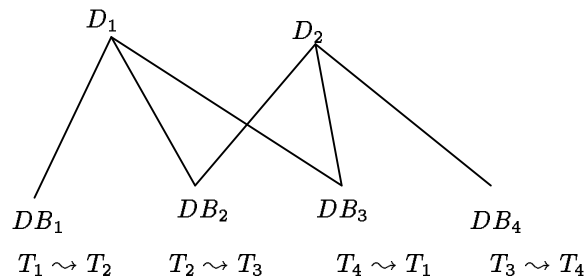

Figure 3: Example of a Non-serializable Execution

protocol to ensure serializability of $\widehat{S}^D$.

5 Ensuring Global Serializability

In the previous section, we developed a mechanism that the domain managers can use to ensure that the projection of the schedule to their domains is serializable. Our mechanism, however, may not ensure global serializability. To see this, let us consider the following example.

Example 3: Consider an MDBS environment consisting of local databases: DBMS_1 with data item x, DBMS_2 with data item z, DBMS_3 with data item y, and DBMS_4 with data item u. Let the domain ordering relation be as illustrated in Figure 3. The set of domains $\Delta = \{DB_1, DB_2, DB_3, DB_4, D_1, D_2\}$, where $D_1 = \bigcup\{DB_1, DB_2, DB_3\}$, and $D_2 = \bigcup\{DB_2, DB_3, DB_4\}$. Consider the following transactions T_1, T_2, T_3, and T_4:

$$\begin{array}{lllllll}
T_1: & b_{11} & w_{11}(x) & b_{13} & w_{13}(y) & c_{11} & c_{13} \\
T_2: & b_{21} & w_{21}(x) & b_{22} & w_{22}(z) & c_{21} & c_{22} \\
T_3: & b_{32} & w_{32}(z) & b_{34} & w_{34}(u) & c_{32} & c_{34} \\
T_4: & b_{44} & w_{44}(u) & b_{43} & w_{43}(y) & c_{44} & c_{43}
\end{array}$$

Note that $Dom(T_1) = D_1$, $Dom(T_2) = D_1$, $Dom(T_3) = D_2$ and $Dom(T_4) = D_2$. Suppose that each local DBMS follows a timestamp scheme for concurrency control in which a timestamp is assigned to a transaction when it begins execution. Since each local DBMS follows the timestamp scheme and the timestamp is assigned to a transaction when it begins execution, the serialization function for a transaction with respect to DB_i, $i = 1, 2, 3, 4$, is the transaction's begin operation at the local DBMSs. Thus, the transactions $\widehat{T_i}$ for the transactions T_1, T_2, T_3, T_4 with respect to each of the domains D_1 and D_2 are as follows:

$$\begin{array}{llllllllllll}
\widehat{T}_1^{D_1}: & b_{11} \quad b_{13} & & \widehat{T}_2^{D_1}: & b_{21} \quad b_{22} & & \widehat{T}_3^{D_1}: & b_{32} & & \widehat{T}_4^{D_1}: & b_{43} \\
\widehat{T}_1^{D_2}: & b_{13} & & \widehat{T}_2^{D_2}: & b_{22} & & \widehat{T}_3^{D_2}: & b_{32} \quad b_{34} & & \widehat{T}_4^{D_2}: & b_{44} \quad b_{43}
\end{array}$$

Consider a schedule S resulting from the concurrent execution of transactions T_1, T_2, T_3, and T_4 such that the local schedules at DBMS_1, DBMS_2, DBMS_3 and DBMS_4 are as follows:

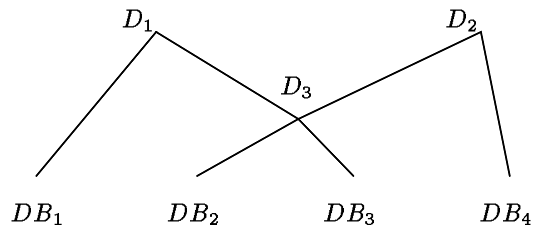

Figure 4: Example of Δ that Satisfies **R1**

$$
\begin{array}{lllllll}
S_1: & b_{11} & w_{11}(x) & b_{21} & w_{21}(x) & c_{11} & c_{21} \\
S_2: & b_{22} & w_{22}(z) & b_{32} & w_{32}(z) & c_{22} & c_{32} \\
S_3: & b_{43} & w_{43}(y) & b_{13} & w_{13}(y) & c_{43} & c_{13} \\
S_4: & b_{34} & w_{34}(u) & b_{44} & w_{44}(y) & c_{34} & c_{44}
\end{array}
$$

Furthermore let the schedules $\widehat{S}^{D_1}$ and $\widehat{S}^{D_2}$ be as follows:

$$
\begin{array}{lllllll}
\widehat{S}^{D_1}: & b_{22} & b_{43} & b_{11} & b_{32} & b_{21} & b_{13} \\
\widehat{S}^{D_2}: & b_{22} & b_{43} & b_{34} & b_{32} & b_{13} & b_{44}
\end{array}
$$

In schedule $\widehat{S}^{D_1}$ operations b_{11}, b_{21}, operations b_{22}, b_{32}, and operations b_{43}, b_{41} conflict. Thus, $\widehat{S}^{D_1}$ is serializable in the order $\widehat{T}_4^{D_1}$, $\widehat{T}_1^{D_1}$, $\widehat{T}_2^{D_1}$, $\widehat{T}_3^{D_1}$. Similarly, in the schedule $\widehat{S}^{D_2}$ operations b_{22}, b_{32}, operations b_{43}, b_{13}, and operations b_{34}, b_{44} conflict. Thus, $\widehat{S}^{D_2}$ is serializable in the order $\widehat{T}_2^{D_2}$, $\widehat{T}_3^{D_2}$, $\widehat{T}_4^{D_2}$, $\widehat{T}_1^{D_2}$. Thus, each schedule $\widehat{S}^{D_1}$, $\widehat{S}^{D_2}$ and $\widehat{S}^{D_3}$ is serializable. However, the global schedule S is not serializable. □

The above example illustrates that even if the domain managers of each domain D ensures that the schedule $\widehat{S}^{D}$ is serializable, the resulting global schedules may not be serializable. For the schedule S to be globally serializable, the set of domains Δ must be appropriately restricted. In the remainder of the section, we consider a restriction on Δ that guarantees that if each domain manager ensures serializability of $\widehat{S}^{D}$, then the resulting global schedule is serializable.

To identify the appropriate restriction on Δ, let us reexamine the non-serializable execution in Example 3. Let the domain managers of the domains D_1 and D_2 ensure serializability of $\widehat{S}^{D_1}$ and $\widehat{S}^{D_2}$ respectively, by following a timestamp scheme in which timestamps are assigned to a transaction when it begins execution. In the schedule $\widehat{S}^{D_1}$, the begin operation for the transactions $\widehat{T}_1^{D_1}$ and $\widehat{T}_3^{D_1}$ are the operations b_{11} and b_{32} respectively. Further, in the schedule $\widehat{S}^{D_2}$, the begin operation for the transactions $\widehat{T}_1^{D_2}$ and $\widehat{T}_3^{D_2}$ are the operations b_{13} and b_{34} respectively. It is possible that the domain manager of the domain D_1 assigns a timestamp to the transaction $\widehat{T}_1^{D_1}$ that is lower than the timestamp it assigns to the transaction $\widehat{T}_3^{D_1}$. In contrast, the domain manager of the domain D_2 assigns a lower timestamp to the transaction $\widehat{T}_3^{D_2}$ than the timestamp it assigns to the transaction $\widehat{T}_3^{D_2}$, thereby resulting in the loss of serializability. If, however, there existed a domain $D_3 = \bigcup\{DB_2, DB_3\}$ (illustrated in Figure 4), then the order in which the domain manager for domain D_1 assigns timestamps to any pair of transactions $\widehat{T}_i^{D_1}$ and $\widehat{T}_j^{D_1}$, and the order in which the domain manager of D_2 assigns timestamps to $\widehat{T}_i^{D_2}$ and $\widehat{T}_j^{D_2}$ must be the same (identical to the order in which D_3 assigns timestamps to transactions, assuming D_3 also follows a timestamping scheme). Hence, if there existed a domain $D_3 = \bigcup\{DB_2, DB_3\}$, then the non-serializable execution in Example 3 would not result. We therefore consider the following restriction on the set Δ of domains:

R1: *For all domains $D_i, D_j \in TOP$, there exists a $D_k \in \Delta$, such that $D_k = D_i \cap D_j$.*

In the domain ordering relation illustrated in Figure 3, since $DB_2 \sqsubset D_1$, $DB_2 \sqsubset D_2$, and $DB_3 \sqsubset D_1$, $DB_3 \sqsubset D_2$, the domain $D_1 \cap D_2$ does not exist. Thus, the corresponding set Δ does not satisfy **R1**. In contrast, in the domain ordering relation illustrated in Figure 4, the domain $D_3 = D_1 \cap D_2$. Thus, the corresponding set Δ satisfies the restriction **R1**.

Unfortunately, even if the set of domain Δ satisfies the restriction **R1**, and each domain manager ensures serializability of the schedule $\widehat{S}^D$, the resulting global schedule may not be serializable. To see this let us consider the following example.

Example 4: Consider an MDBS environment consisting of local databases: DBMS_1 with data item x, DBMS_2 with data item y, and DBMS_3 with data item z. Let the domain ordering be as illustrated in Figure 5. The set of domains $\Delta = \{DB_1, DB_2, DB_3, D_1, D_2, D_3\}$, where $D_1 = \bigcup\{DB_1, DB_2\}$, $D_2 = \bigcup\{DB_2, DB_3\}$, and $D_3 = \bigcup\{DB_1, DB_3\}$. Further, the set $TOP = \{D_1, D_2, D_3\}$, $D_1 \cap D_2 = DB_2$, $D_2 \cap D_3 = DB_3$, and $D_1 \cap D_3 = DB_1$. Hence, Δ satisfies the restriction **R1**. Consider the following transactions T_1, T_2, and T_3 that execute:

$$
\begin{array}{lllllll}
T_1: & b_{11} & w_{11}(x) & b_{13} & w_{13}(y) & c_{11} & c_{13} \\
T_2: & b_{21} & w_{21}(x) & b_{22} & w_{22}(z) & c_{21} & c_{22} \\
T_3: & b_{32} & w_{32}(z) & b_{33} & w_{33}(y) & c_{32} & c_{34}
\end{array}
$$

Note that $Dom(T_1) = D_3$, $Dom(T_2) = D_1$, and $Dom(T_3) = D_2$. Suppose that each local DBMS follows a timestamp scheme for concurrency control in which a timestamp is assigned to a transaction when it begins execution. Since each local DBMS follows the TO scheme and the timestamps are assigned to transactions when they begin execution, the serialization function for a transaction with respect to DB_i, $i = 1, 2, 3, 4$, is the transactions' begin operation at the local DBMSs. Thus, the transactions $\widehat{T}_i$ for the transactions T_1, T_2, T_3 with respect to each of the domains D_1, D_2 and D_3 are as follows:

$$
\begin{array}{lll}
\widehat{T}_1^{D_1}: \; b_{11} & \widehat{T}_2^{D_1}: \; b_{21} \;\; b_{22} & \widehat{T}_3^{D_1}: \; b_{32} \\
\widehat{T}_1^{D_2}: \; b_{13} & \widehat{T}_2^{D_2}: \; b_{22} & \widehat{T}_3^{D_2}: \; b_{32} \;\; b_{33} \\
\widehat{T}_1^{D_3}: \; b_{11} \;\; b_{13} & \widehat{T}_2^{D_3}: \; b_{21} & \widehat{T}_3^{D_3}: \; b_{33}
\end{array}
$$

Consider a schedule S resulting from the concurrent execution of transactions T_1, T_2, and T_3 such that the local schedules at DBMS_1, DBMS_2, DBMS_3 and DBMS_4 are as follows:

$$
\begin{array}{lllllll}
S_1: & b_{11} & w_{11}(x) & b_{21} & w_{21}(x) & c_{11} & c_{21} \\
S_2: & b_{22} & w_{22}(z) & b_{32} & w_{32}(z) & c_{22} & c_{32} \\
S_3: & b_{33} & w_{33}(y) & b_{13} & w_{13}(y) & c_{33} & c_{13}
\end{array}
$$

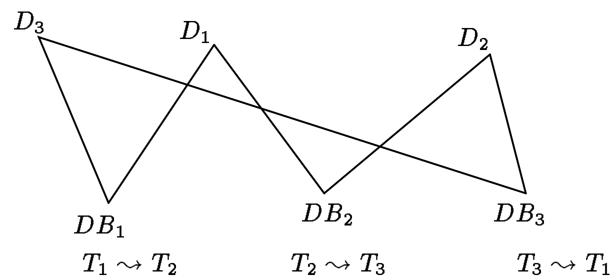

Figure 5: A Domain Ordering with a cyclic DG

Furthermore let the schedules $\widehat{S}^{D_1}$, $\widehat{S}^{D_2}$ and $\widehat{S}^{D_3}$ be as follows:

$$\begin{array}{lllll} \widehat{S}^{D_1}: & b_{11} & b_{22} & b_{21} & b_{32} \\ \widehat{S}^{D_2}: & b_{22} & b_{33} & b_{32} & b_{13} \\ \widehat{S}^{D_3}: & b_{11} & b_{33} & b_{21} & b_{13} \end{array}$$

In schedule $\widehat{S}^{D_1}$ operations b_{11}, b_{21}, and operations b_{22}, b_{32}, conflict. Thus, $\widehat{S}^{D_1}$ is serializable in the order $\widehat{T}_1^{D_1}$, $\widehat{T}_2^{D_1}$, $\widehat{T}_3^{D_1}$. In the schedule $\widehat{S}^{D_2}$ operations b_{22}, b_{32}, and operations b_{33}, b_{13} conflict. Thus, $\widehat{S}^{D_2}$ is serializable in the order $\widehat{T}_2^{D_2}$, $\widehat{T}_3^{D_2}$, $\widehat{T}_1^{D_2}$. Similarly, in the schedule $\widehat{S}^{D_3}$ operations b_{11}, b_{21}, and operations b_{33}, b_{13} conflict. Thus, $\widehat{S}^{D_3}$ is serializable in the order $\widehat{T}_3^{D_3}$, $\widehat{T}_1^{D_3}$, $\widehat{T}_2^{D_3}$. Thus, each schedule $\widehat{S}^{D_1}$, $\widehat{S}^{D_2}$ and $\widehat{S}^{D_3}$ is serializable. However, the global schedule S is not serializable. □

The above example illustrates that even if Δ satisfies the restriction **R1**, ensuring serializability of $\widehat{S}^D$ for each domain D may not ensure global serializability. To identify conditions under which global serializability is ensured we need to introduce the notion of a *domain graph.* A domain graph (DG) for a set of domains Δ, is an undirected graph whose nodes correspond to the set of domains $D \in TOP$. Let D_i and D_j be two nodes in DG. There is an edge (D_i, D_j) in DG if there exists a domain $D_k \in \Delta$ such that $D_k \sqsubset D_i$ and $D_k \sqsubset D_j$.

Theorem 3: Consider an MDBS environment with the set Δ of domains. Let S be a global schedule. Further, let each of the following three hold:

- For each DB_k such that $DB_k \sqsubset D$, S_k is serializable and further there exists a function ser_{S_k} such that for all transactions T_i, T_j, $global(T_i, DB_k)$, $global(T_j, DB_k)$, and $T_i \overset{*}{\rightsquigarrow}_{S_k} T_j$, then $ser_{S_k}(T_i) \prec_S ser_{S_k}(T_j)$.
- For all domains $D \in \Delta$ such that $D \notin TOP$, $\widehat{S}^D$ is serializable and further there exists a function $ser_{\widehat{S}^D}$ such that for all transactions T_i, T_j, if $global(T_i, D)$, $global(T_j, D)$, and $\widehat{T}_i^D \overset{*}{\rightsquigarrow}_{\widehat{S}^D}$ $\widehat{T}_j^D$, then $ser_{\widehat{S}^D}(\widehat{T}_i^D) \prec_S ser_{\widehat{S}^D}(\widehat{T}_j^D)$.
- For all domains $D \in \Delta$ such that $D \in TOP$, $\widehat{S}^D$ is serializable.

If Δ satisfies **R1** and the DG is acyclic, then S is serializable. $\Box$

The DG for the set of domains Δ corresponding to the domain ordering relation illustrated in Figure 4 contains nodes D_1 and D_2 and an edge (D_1, D_2). Since this DG is acyclic and the set of domains Δ satisfies **R1**, it follows that in order to ensure global serializability, it suffices to ensure that the schedules $\widehat{S}^D$, for each domain $D \in \Delta$, is serializable. In contrast, the DG for the set of domains corresponding to the domain ordering relation illustrated in Figure 5 contains a cycle (D_1, D_2), (D_2, D_3) and (D_3, D_1). Hence, even if for each domain $D \in \Delta$, the schedule $\widehat{S}^D$ is serializable and the set of domains Δ satisfies restriction **R1**, loss of global serializability may result.

In the superdatabase architecture, proposed in [Pu88], the set of domains Δ is restricted as follows:

For all domains D_i, D_j, if $child(D_i, D_j)$, then for all $D_k \neq D_j$, $\neg child(D_i, D_k)$.

It is easy to see that this is a special instance of Δ that satisfies restriction **R1** and, further, the domain graph corresponding to Δ is acyclic. Thus, from Theorem 3, it follows that a concurrency control scheme based on ensuring the serializability of $\widehat{S}^D$ for each domain $D \in \Delta$ can be used in superdatabases to ensure global serializability.

In contrast to our scheme, where for each domain D, the domain manager $DM(D)$ ensures serializability of the schedule $\widehat{S}^D$, [Pu88] uses a protocol referred to as the *hierarchical validation*, in order to ensure global serializability. The following two differences between our approach and the hierarchical validation protocol are noteworthy. First, in hierarchical validation, each domain manager must follow a SGT certification based protocol. In contrast, in our approach, different domain managers may follow different concurrency control protocols (centralized or distributed), for ensuring serializability of the schedule $\widehat{S}^D$. Second, in our approach, for a transaction T_i and a domain D such that $local(T_i, D)$, the domain manager of D does *not* need to submit any operations of T_i to the parent domain of D. In contrast, in the hierarchical validation protocol, the domain manager for the domain D must submit operations of all the transactions, whether $local(T_i, D)$ or $global(T_i, D)$, to the parent domain of D. A detailed comparison between our approach and the hierarchical validation protocol suggested in [Pu88] can be found in Appendix A.

Example 4 illustrates that if the DG contains a cycle, our scheme may not ensure global serializability. However, not every cycle in the DG would result in a potential loss of serializability. Consider, for example, DG for the set of domains corresponding to the domain ordering relation illustrated in Figure 6. Note that DG contains a cycle (D_2, D_3), (D_3, D_4), (D_4, D_2). However, for the set of domains corresponding to the domain ordering relation illustrated in Figure 6, if for each $D \in \Delta$, the domain manager for D ensures that the schedule $\widehat{S}^D$ is serializable, the resulting global schedule S would be serializable. Thus, certain cycles in DG do not result in a potentially non-serializable global schedule. To identify such cycles, we introduce the notion of the *labeled domain graph* (LDG). An LDG is a domain graph in which each edge (D_i, D_j) has a label, referred to as

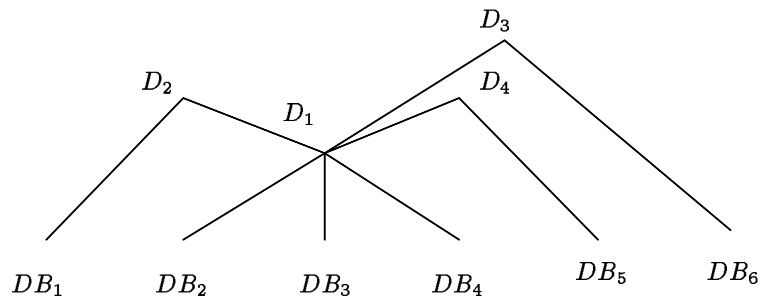

Figure 6: A Domain Ordering such that LDG Contains No Undesirable Cycles

$label(D_i, D_j)$, where $label(D_i, D_j) = D_i \cap D_j$. Let $(D_1, D_2), (D_2, D_3), \ldots, (D_{r-1}, D_r), (D_r, D_1)$ be a cycle in the LDG. We refer to the cycle in the LDG as a *undesirable cycle* iff for all k, l, $k = 1, 2, \ldots, r$, $l = 1, 2, \ldots, r$, if $k \neq l$, then $label(D_k, D_{(k+1) mod\ r}) \neq label(D_l, D_{(l+1) mod\ r})$. Note that the LDG for the set of domains corresponding to the domain ordering relation illustrated in Figure 6, has edges (D_2, D_3), (D_3, D_4) and (D_4, D_2), where $label(D_2, D_3) = label(D_3, D_4) = label(D_4, D_2) = D_1$. Thus, LDG does not contain any undesirable cycles. In contrast, the LDG for the set of domains corresponding to the domain ordering illustrated in Figure 5 contains a cycle (D_1, D_2), (D_2, D_3), (D_3, D_1), where $label(D_1, D_2) = DB_2$, $label(D_2, D_3) = DB_3$, $label(D_3, D_1) = DB_1$. Hence, LDG contains a undesirable cycle. If the LDG for the set of domains Δ does not contain any undesirable cycles, then ensuring that $\widehat{S}^D$, for each domain $D \in \Delta$ would ensure global serializability as is stated in the following theorem.

Theorem 4: Consider an MDBS environment with the set Δ of domains. Let S be a global schedule. Further, let each of the following three hold:

- For each DB_k such that $DB_k \sqsubset D$, S_k is serializable and further there exists a function ser_{S_k} such that for all transactions T_i, T_j, $global(T_i, DB_k)$, $global(T_j, DB_k)$, and $T_i \overset{*}{\leadsto}_{S_k} T_j$, then $ser_{S_k}(T_i) \prec_S ser_{S_k}(T_j)$.
- For all domains $D \in \Delta$, such that $D \notin TOP$, $\widehat{S}^D$ is serializable and further there exists a function $ser_{\widehat{S}^D}$ such that for all transactions T_i, T_j, if $global(T_i, D)$, $global(T_j, D)$, and $\widehat{T}_i^D \overset{*}{\leadsto}_{\widehat{S}^D} \widehat{T}_j^D$, then $ser_{\widehat{S}^D}(\widehat{T}_i^D) \prec_S ser_{\widehat{S}^D}(\widehat{T}_j^D)$.
- For all domains $D \in \Delta$ such that $D \in TOP$, $\widehat{S}^D$ is serializable.

If Δ satisfies **R1** and LDG contains no undesirable cycles, then S is serializable. $\Box$

6 Conclusions

A multidatabase system (MDBS) is a facility, developed on top of pre-existing local database management systems (DBMSs), that provides users of a DBMS access and update privileges to data located in other heterogeneous data sources. Over the past decade, substantial research has been done to identify mechanisms for effectively dealing with the problems that arise due to the *heterogeneity* and *autonomy* of the local systems. This research has resulted in transaction management algorithms for MDBSs that ensure correctness without sacrificing the autonomy of the individual system. Most of the proposed approaches have, however, considered an MDBS as a single monolithic system which, executing on top of the existing local DBMSs, controls the execution and commitment of the *global transactions* (transactions that execute at multiple local DBMSs) in such a way that consistency of the individual systems is not jeopardized.

One issue that has been relatively ignored is that of the architecture of MDBSs. We believe that a large MDBS, that span multiple organizations distributed over various geographically distant locations, will not be developed as a single monolithic system; rather, it will be developed hierarchically. However, if the architecture of the MDBS is hierarchical, then the transaction management algorithms followed by individual MDBSs needs to be *composable* in such a way that it is feasible to incorporate individual MDBSs as elements in a larger MDBS. In this paper, we presented a hierarchical architecture for MDBSs. In our architecture, with an MDBS environment is associated a set of *domain* Δ with an ordering relation $\sqsubset$. A domain is either a set of data items at some local DBMS, or it may consist of a union of the set of data items in other domains. The execution of the transactions within a domain is controlled by the *domain manager*. We adopted serializability as the correctness criteria and developed a mechanism which the domain managers can use to ensure that the concurrent execution of the transactions does not result in a loss of serializability within their domains. Furthermore, we identified restrictions on the domain order that ensure global serializability.

In this paper, we did not consider the issue of failure-resilience. Failure-resilience in MDBSs is complicated since the requirement of autonomy preservation renders the usage of *atomic commit protocols* [BHG87] unsuitable for MDBS environments. In the absence of atomic commit protocols, it is possible that certain subtransactions of a multi-site transaction commit, whereas others abort, thereby violating the atomicity property. The problem of ensuring atomicity in MDBS environments has been studied in [BST90, WV90, VW90, MRB$^+$92b, MRKS92]. We need to further study how these schemes can be adapted for our MDBS architecture.

Finally, in this paper we concentrated only on developing mechanisms for ensuring global serializability in MDBSs that conform to our architecture. Since ensuring global serializability in an MDBS environment is both complex and expensive, substantial research has been done to develop correctness criteria for MDBSs that are weaker than serializability but that ensure database consistency under appropriate assumptions about the MDBS environment [DE89, MRKS91]. It will be interesting to study concurrency control schemes and the consistency guarantee that results in MDBSs in which different domains may follow different notions of correctness.

7 Acknowledgements

We wish to thank Daniel Barbará for many inspiring discussions. We would further like to thank Rajeev Rastogi for his comments on an earlier draft of the paper.

References

[BGR92] R.K. Batra, D. Georgakopoulos, and M. Rusinkiewicz. A decentralized deadlock-free concurrency control method for multidatabase transactions. In *Proceedings of the Twelfth International Conference on Distributed Computing Systems, Yokohoma, Japan*, 1992.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Reading, MA, 1987.

[BS88] Y. Breitbart and A. Silberschatz. Multidatabase update issues. In *Proceedings of ACM-SIGMOD 1988 International Conference on Management of Data, Chicago*, pages 135–141, 1988.

[BST90] Y. Breitbart, A. Silberschatz, and G. R. Thompson. Reliable transaction management in a multidatabase system. In *Proceedings of ACM-SIGMOD 1990 International Conference on Management of Data, Atlantic City, New Jersey*, pages 215–224, 1990.

[DE89] W. Du and A. K. Elmagarmid. Quasi serializability: a correctness criterion for global concurrency control in InterBase. In *Proceedings of the Fifteenth International Conference on Very Large Databases, Amsterdam*, pages 347–355, 1989.

[ED90] A.K. Elmagarmid and W. Du. A paradigm for concurrency control in heterogeneous distributed database systems. In *Proceedings of the Sixth International Conference on Data Engineering, Los Angeles*, 1990.

[GRS91] D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth. On serializability of multidatabase transactions through forced local conflicts. In *Proceedings of the Seventh International Conference on Data Engineering, Kobe, Japan*, 1991.

[MRB+92a] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth, and A. Silberschatz. The concurrency control problem in multidatabases: Characteristics and solutions. In *Proceedings of ACM-SIGMOD 1992 International Conference on Management of Data, San Diego, California*, 1992.

[MRB+92b] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth, and A. Silberschatz. Ensuring transaction atomicity in multidatabase systems. In *Proceedings of the Eleventh*

ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, California, 1992.

[MRKS91] S. Mehrotra, R. Rastogi, H. F. Korth, and A. Silberschatz. Non-serializable executions in heterogeneous distributed database systems. In *Proceedings of the First International Conference on Parallel and Distibuted Information Systems, Miami Beach, Florida*, 1991.

[MRKS92] S. Mehrotra, R. Rastogi, H. F. Korth, and A. Silberschatz. A transaction model for heterogeneous distributed database systems. In *Proceedings of the Twelfth International Conference on Distributed Computing Systems, Yokohoma, Japan*, 1992.

[Pu88] C. Pu. Superdatabases for composition of heterogeneous databases. In *Proceedings of the Fourth International Conference on Data Engineering, Los Angeles*, 1988.

[VW90] J. Veijalainen and A. Wolski. The 2PC agent method and its correctness. Technical Report Research Notes 1192, Technical research Centre of Finland, December 1990.

[WV90] A. Wolski and J. Veijalainen. 2PC agent method: Achieving serializability in presence of failures in a heterogeneous multidatabase. In *Proceedings of the International conference on databases, parallel architectures and their applications*, pages 321–330, March 1990.

A Comparison With Superdatabases

In this section, we compare our scheme with the hierarchical validation protocol suggested for superdatabases in [Pu88]. Before we do so, we first describe the hierarchical validation protocol.

In the hierarchical validation protocol, the domain manager for a domain D, $D \notin TOP$, for each transaction T_i such that $\neg local(T_i, DB_k)$, $k = 1, 2, \ldots, m$, (that is, transactions that are not local to any local DBMS), submits the operations $\{sf(T_i, DB_k) \mid DB_k \sqsubseteq D$ and T_i accesses data item in $DB_k\}$ to the domain manager of the domain D', where $parent(D', D)$[4]. We refer to the restriction of a transaction T_i to the set of operation $\{sf(T_i, DB_k) \mid DB_k \sqsubseteq D$ and T_i accesses data item in $DB_k\}$ as a transaction $\bar{T}_i^D$. Further, we refer to the restriction of the schedule S to the operations belonging to transactions $\bar{T}_i^D$ by $\bar{S}^D$. In the schedule $\bar{S}^D$ operations $sf(T_i, DB_k)$ and $sf(T_j, DB_l)$, $i \neq j$, are defined to conflict iff $k = l$. In the hierarchical validation protocol, for each domain $D \neq DB_k$, $k = 1, 2, \ldots, m$, the domain manager $DM(D)$ follows the SGT certification protocol to ensure that the schedule $\bar{S}^D$ is serializable.

The following two differences between our approach and the hierarchical validation protocol are noteworthy. First, in hierarchical validation, each domain manager follows the SGT certification protocol to ensure serializability of $\bar{S}^D$. In contrast, in our approach, different domain managers may follow different concurrency control protocols (centralized or distributed), for ensuring serializability of the schedule $\widehat{S}^D$. Second, in our approach, for a transaction T_i and a domain D such that $local(T_i, D)$, the domain manager of D does not submit any information to the parent domain of D. In contrast, in the hierarchical validation protocol, the $DM(D)$ must submit the set of operations $\{sf(T_i, DB_k) \mid DB_k \sqsubseteq D$ and T_i accesses data item in $DB_k\}$ for each transaction T_i, whether $local(T_i, D)$ or $global(T_i, D)$, to the parent domain of D. If, in the hierarchical validation protocol, $DM(D)$ does not submit the operations $sf(T_i, D_k)$, where $local(T_i, D)$, to the parent domain of D, then the protocol may not ensure global serializability. We illustrate this in the following example.

Example 5: Consider an MDBS environment consisting of local databases: DBMS_1 with data item x, DBMS_2 with data item y, DBMS_3 with data item z, and DBMS_4 with data item u. Let the set of domains $\Delta = \{DB_1, DB_2, DB_3, DB_4, D_1, D_2, D_3\}$, where $D_1 = \bigcup\{DB_1, DB_2\}$, $D_2 = \{DB_3, DB_4\}$, and $D_3 = \bigcup\{D_1, D_2\}$. Note that the set of domains Δ conforms to the superdatabase architecture. Consider the following transactions T_1, T_2, T_3 and T_4 that execute:

$$
\begin{array}{lllllll}
T_1: & b_{11} & w_{11}(x) & b_{13} & w_{13}(z) & c_{11} & c_{13} \\
T_2: & b_{22} & w_{22}(y) & b_{24} & w_{24}(u) & c_{22} & c_{24} \\
T_3: & b_{31} & w_{32}(x) & b_{32} & w_{32}(y) & c_{32} & c_{32} \\
T_4: & b_{43} & w_{43}(z) & b_{44} & w_{44}(u) & c_{43} & c_{44}
\end{array}
$$

Note that $Dom(T_1) = D_3$, $Dom(T_2) = D_3$, $Dom(T_3) = D_1$ and $Dom(T_4) = D_2$. Further, $global(T_1, D_1)$, $global(T_2, D_1)$ and $local(T_3, D_1)$. Similarly, $global(T_1, D_2)$, $global(T_2, D_2)$ and $local(T_4, D_2)$.

[4] Note that in the superdatabases each domain may have at most one *parent*.

Suppose that each local DBMS follows a timestamp scheme for concurrency control in which a timestamp is assigned to a transaction when it begins execution. Since each local DBMS follows the timestamp scheme and the timestamps is assigned to a transaction when it begins execution, the serialization function for a transaction with respect to DB_i, $i = 1, 2, 3, 4$, is the transactions' begin operation at the local DBMSs. Thus, the transactions $\bar{T}_i^D$ for the transactions T_1, T_2, T_3, T_4 with respect to each of the domains D_1, D_2 and D_3 are as follows:

$\bar{T}_1^{D_1}$: b_{11} $\quad$ $\bar{T}_2^{D_1}$: b_{22} $\quad$ $\bar{T}_3^{D_1}$: b_{31} b_{32}

$\bar{T}_1^{D_2}$: b_{13} $\quad$ $\bar{T}_2^{D_2}$: b_{24} $\quad$ $\bar{T}_4^{D_2}$: b_{43} b_{44}

$\bar{T}_1^{D_3}$: b_{11} b_{13} $\quad$ $\bar{T}_2^{D_3}$: b_{22} b_{24}

Consider a schedule S resulting from the concurrent execution of transactions T_1, T_2, T_4, and T_4 such that the local schedules at DBMS$_1$, DBMS$_2$, DBMS$_3$ and DBMS$_4$ are as follows:

$$
\begin{array}{lllllll}
S_1: & b_{11} & w_{11}(x) & b_{31} & w_{31}(x) & c_{11} & c_{31} \\
S_2: & b_{32} & w_{32}(y) & b_{22} & w_{22}(y) & c_{32} & c_{22} \\
S_3: & b_{43} & w_{43}(z) & b_{13} & w_{13}(z) & c_{43} & c_{13} \\
S_4: & b_{24} & w_{24}(u) & b_{44} & w_{44}(u) & c_{24} & c_{44}
\end{array}
$$

Let us assume that since $local(T_3, D_1)$ and $local(T_4, D_2)$, the schedule $\bar{S}^{D_3}$ does not contain the transactions $\bar{T}_3^{D_3}$ and $\bar{T}_4^{D_3}$. Thus, the schedule $\bar{S}^D$ are as follows:

$\bar{S}^{D_1}$: b_{11} b_{31} b_{32} b_{22}

$\bar{S}^{D_2}$: b_{24} b_{44} b_{43} b_{13}

$\bar{S}^{D_3}$: b_{11} b_{24} b_{13} b_{22}

In schedule $\bar{S}^{D_1}$ operations b_{11}, b_{31}, and operations b_{32}, b_{22}, conflict. Thus, $\bar{S}^{D_1}$ is serializable in the order $\bar{T}_1^{D_1}$, $\bar{T}_3^{D_1}$, $\bar{T}_2^{D_1}$. In the schedule $\bar{S}^{D_2}$ operations b_{24}, b_{44}, and operations b_{43}, b_{13} conflict. Thus, $\bar{S}^{D_2}$ is serializable in the order $\bar{T}_2^{D_2}$, $\bar{T}_4^{D_2}$, $\bar{T}_1^{D_2}$. In the schedule $\bar{S}^{D_3}$ no two operations conflict. Thus, $\bar{S}^{D_3}$ is serializable. Note that since each schedule $\bar{S}^D$, $D = D_1, D_2, D_3$ is serializable, it could have been generated by the hierarchical validation protocol. However, the global schedule S is not serializable. Thus, for the hierarchical validation protocol to ensure global serializability, the schedule $\bar{S}^{D_3}$ must contain transactions $\bar{T}_3^{D_3}$ and $\bar{T}_4^{D_3}$. Hence, the domain managers of domains D_1 and D_2 must submit the operations belonging to the transactions $\bar{T}_3^{D_3}$ and $\bar{T}_4^{D_3}$ to the domain manager of D_3. □

B Proofs of the Theorems

Proof of Lemma 1: Assume that S^D is not serializable. Since by (1) each S^{D_k} is serializable, there exist transactions $T_1, T_2, \ldots, T_n$ such that $T_1 \overset{*}{\leadsto}_{S^{D_{k_1}}} T_2$, $T_2 \overset{*}{\leadsto}_{S^{D_{k_2}}} T_3$, …, $T_{n-1} \overset{*}{\leadsto}_{S^{D_{k_{n-1}}}} T_n$, $T_n \overset{*}{\leadsto}_{S^{D_{k_n}}} T_1$, where $child(D_{k_i}, D)$, $global(T_i, D_{k_i})$, and $global(T_{(i+1) mod\ n}, D_{k_i})$, $i = 1, 2, \ldots, n$. By (2), $sf(T_1, D_{k_1}) \prec_S sf(T_2, D_{k_1})$, $sf(T_2, D_{k_2}) \prec_S sf(T_3, D_{k_2})$, …, $sf(T_{n-1}, D_{k_{n-1}}) \prec_S sf(T_n, D_{k_{n-1}})$, $sf(T_n, D_{k_n}) \prec_S sf(T_1, D_{k_n})$. Thus, by the definition of conflicts in $\widehat{S}^D$, $\widehat{T}_1^D \leadsto_{\widehat{S}^D} \widehat{T}_2^D$, $\widehat{T}_2^D \leadsto_{\widehat{S}^D} \widehat{T}_3^D$, …, $\widehat{T}_{n-1}^D \leadsto_{\widehat{S}^D} \widehat{T}_n^D$, $\widehat{T}_n^D \leadsto_{\widehat{S}^D} \widehat{T}_1^D$. Hence, $\widehat{T}_1^D \overset{*}{\leadsto}_{\widehat{S}^D} \widehat{T}_1^D$ which is a contradiction since $\widehat{S}^D$ by (3) above is serializable. Hence proved. □

In order to prove Lemma 2, we need to associate a notion of a level with a domain:

$$level(D) = \begin{cases} 1 \text{ if } D = DB_k \text{ for some local database DBMS}_k \\ maximum(level(D_k)) + 1 \quad \text{where } child(D_k, D) \end{cases} \quad \square$$

Proof of Lemma 2: The proof is by the induction over the level of the domains.

Basis ($level(D) = 1$): If $level(D) = 1$, then for some DB_k, $D = DB_k$. Hence, for all transactions T_i, T_j, if $global(T_i, D)$, $global(T_j, D)$, and $T_i \overset{*}{\leadsto}_{S_k} T_j$, then by definition of ser_{S_k}, $ser_{S_k}(T_i) \prec_S ser_{S_k}(T_j)$. Hence, $sf(T_i, D) \prec_S sf(T_j, D)$.

Induction: Assume that the lemma is true for all domains D such that $level(D) \leq p$. Let $D = \bigcup\{D_1, D_2, \ldots, D_n\}$ be an arbitrary domain such that $level(D) = p + 1$. Let T_i, T_j be transactions such that $global(T_i, D)$, $global(T_j, D)$, and $T_i \overset{*}{\leadsto}_{S^D} T_j$. There are two cases to consider:

- ($T_i \overset{*}{\leadsto}_{S^{D_k}} T_j$ for some D_k such that $child(D_k, D)$:) Since $global(T_i, D)$ and $global(T_j, D)$ and T_i, T_j executes in D_k, $global(T_i, D_k)$ and $global(T_j, D_k)$. Thus, by IH, $sf(T_i, D_k) \prec_S sf(T_j, D_k)$. Hence, by definition of a conflict in $\widehat{S}^D$, $\widehat{T}_i^D \leadsto_{\widehat{S}^D} \widehat{T}_j^D$. As a result, by the definition of $sf(T, D)$, $sf(T_i, D) \prec_S sf(T_j, D)$

- (There exist transactions $T_1, T_2, \ldots, T_n$ such that $T_i \overset{*}{\leadsto}_{S^{D_{k_1}}} T_1$, $T_1 \overset{*}{\leadsto}_{S^{D_{k_2}}} T_2$, …, $T_{n-1} \overset{*}{\leadsto}_{S^{D_{k_{n-1}}}} T_n$, $T_n \overset{*}{\leadsto}_{S^{D_{k_n}}} T_j$, where $child(D_{k_i}, D)$, $i = 1, 2, \ldots, n$:) Note that $global(T_i, D_{k_i})$, $i = 1, 2, \ldots, n$, and $global(T_{i+1}, D_{k_i})$, $i = 1, 2, \ldots, n-1$. Thus, by IH, $sf(T_i, D_{k_1}) \prec_S sf(T_1, D_{k_1})$, $sf(T_1, D_{k_2}) \prec_S sf(T_2, D_{k_2})$, …, $sf(T_{n-1}, D_{k_{n-1}}) \prec_S sf(T_n, D_{k_{n-1}})$, $sf(T_n, D_{k_n}) \prec_S sf(T_j, D_{k_n})$. Hence, by definition of a conflict in $\widehat{S}^D$, $\widehat{T}_i^D \leadsto_{\widehat{S}^D} \widehat{T}_1^D$, $\widehat{T}_1^D \leadsto_{\widehat{S}^D} \widehat{T}_2^D$, …, $\widehat{T}_{n-1}^D \leadsto_{\widehat{S}^D} \widehat{T}_n^D$, $\widehat{T}_n^D \leadsto_{\widehat{S}^D} \widehat{T}_j^D$. Hence, $\widehat{T}_i^D \overset{*}{\leadsto}_{\widehat{S}^D} \widehat{T}_j^D$. As a result, by the definition of $sf(T, D)$, $sf(T_i, D) \prec_S sf(T_j, D)$. Hence proved. □

Proof of Theorem 2: The proof is by the induction over the level of the domain D.

Basis ($level(D) = 1$): If $level(D) = 1$, then for some DB_k, $D = DB_k$. Since S_k is serializable, for all $k = 1, 2, \ldots, n$, S^D is serializable.

Induction: Assume that the theorem is true for each D such that $level(D) \leq p$. We show it to be true for each domain, D such that $level(D) = p + 1$. Let D be such a domain and further let $D = \bigcup\{D_1, D_2, \ldots, D_n\}$. Since $level(D_k) \leq p$, $child(D_k, D)$, by IH, S^{D_k} is serializable. Further, since

$child(D_k, D)$, $D_k \notin TOP$. Thus, the function $ser_{\widehat{S}^{D_k}}$ exists. By Lemma 2, $sf(T_i, D_k) = ser_{\widehat{S}^{D_k}}(T_i)$ satisfies the property that for all T_i, T_j, such that $global(T_i, D_k)$, $global(T_j, D_k)$, $T_i \overset{*}{\rightsquigarrow}_{S^{D_k}} T_j \Rightarrow sf(T_i, D_k) \prec_S sf(T_j, D_k)$. Thus, by Lemma 1, since $\widehat{S}^D$ is serializable, S^D is serializable. Hence proved. □

Proof of Theorem 3 and 4: Theorem 3 directly follows from Theorem 4. So we only prove Theorem 4. To prove Theorem 4, we will need the following lemmas.

Lemma 3: Let D be a domain and T_i, T_j be transactions such that $global(T_i, D)$ and $global(T_j, D)$. If there exists a $D' \sqsubseteq D$ such that $\widehat{T}_i^{D'} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D'}} \widehat{T}_j^{D'}$, then $\widehat{T}_i^{D} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D}} \widehat{T}_j^{D}$.

Proof: The proof in by induction on the level of the domain D, where $D' \sqsubseteq D$.

Basis ($level(D) = level(D')$)**:** Since $D' \sqsubseteq D$, it must be the case that $D' = D$. Thus, $\widehat{T}_i^{D} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D}} \widehat{T}_j^{D}$.

Induction: Assume that the lemma is true for all domains D, $D' \sqsubseteq D$ such that $level(D) \leq level(D') + p$. We show that the lemma is true for all domains such that $level(D) = level(D') + p + 1$. Let D be such a domain. Since $D' \sqsubseteq D$, there exists a domain D'', $D' \sqsubseteq D''$, where $child(D'', D)$. Further, since $global(T_i, D)$ and $global(T_j, D)$, and since T_i and T_j execute in D'', it must be the case that $global(T_i, D'')$ and $global(T_j, D'')$. Thus, by IH, $\widehat{T}_i^{D''} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D''}} \widehat{T}_j^{D''}$. Since $\widehat{T}_i^{D''} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D''}} \widehat{T}_j^{D''}$, by definition of sf, $sf(T_i, D'') \prec_S sf(T_j, D'')$. Thus, by definition of $\widehat{T}_i$, $\widehat{T}_i^{D} \rightsquigarrow_{\widehat{S}^{D}} \widehat{T}_j^{D}$. Hence, $\widehat{T}_i^{D} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D}} \widehat{T}_j^{D}$. □

Lemma 4: Let T_i and T_j be transactions and D be a domain such that $global(T_i, D)$ and $global(T_j, D)$ and $level(D) \geq 2$. If $T_i \overset{*}{\rightsquigarrow}_{S^D} T_j$, then $\widehat{T}_i^{D} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D}} \widehat{T}_j^{D}$.

Proof: Let $p = level(D)$ The proof is by induction on p.

Basis ($p = 2$)**:** Thus, $D = \{DB_1, DB_2, \ldots, DB_m\}$ for some local database DBMS$_k$, $k = 1, 2, \ldots, m$. Since $T_i \overset{*}{\rightsquigarrow}_{S^D} T_j$, there exists transactions $T_1, T_2, \ldots, T_n$ such that $T_i \overset{*}{\rightsquigarrow}_{S_{k_1}} T_1$, $T_1 \overset{*}{\rightsquigarrow}_{S_{k_2}} T_2$, …, $T_{n-1} \overset{*}{\rightsquigarrow}_{S_{k_n}} T_n$, $T_n \overset{*}{\rightsquigarrow}_{S_{k_{n+1}}} T_j$, where $global(T_l, DB_{k_l})$ and $global(T_l, DB_{k_{(l+1)}})$, where $l = 1, 2, \ldots, n$. Hence by definition of the serialization function sf, $sf(T_i, DB_{k_1}) \prec_S sf(T_1, DB_{k_1})$, $sf(T_1, DB_{k_2}) \prec_S sf(T_2, DB_{k_2})$, …, $sf(T_{n-1}, DB_{k_n}) \prec_S sf(T_n, DB_{k_n})$, and $sf(T_n, DB_{k_{n+1}}) \prec_S sf(T_j, DB_{k_{n+1}})$. Thus, by definition of $\widehat{T}_i$, $\widehat{T}_i^{D} \rightsquigarrow_{\widehat{S}^{D}} \widehat{T}_1^{D}$, $\widehat{T}_1^{D} \rightsquigarrow_{\widehat{S}^{D}} \widehat{T}_2^{D}$, …, $\widehat{T}_{n-1}^{D} \rightsquigarrow_{\widehat{S}^{D}} \widehat{T}_n^{D}$, and $\widehat{T}_n^{D} \rightsquigarrow_{\widehat{S}^{D}} \widehat{T}_j^{D}$. Hence, $\widehat{T}_i^{D} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D}} \widehat{T}_j^{D}$.

Induction: Assume that the lemma holds for all domains such that $level(D) \leq p$. We show that it holds for domains such that $level(D) = p + 1$. Let $D = \bigcup\{D_1, D_2, \ldots, D_m\}$ be an arbitrary domain such that $level(D) = p + 1$. Since $T_i \overset{*}{\rightsquigarrow}_{S^D} T_j$, there exists transactions $T_1, T_2, \ldots, T_n$, $n \geq 0$, such that $T_i \overset{*}{\rightsquigarrow}_{S^{D_{k_1}}} T_1$, $T_1 \overset{*}{\rightsquigarrow}_{S^{D_{k_2}}} T_2$, …, $T_{n-1} \overset{*}{\rightsquigarrow}_{S^{D_{k_n}}} T_n$, and $T_n \overset{*}{\rightsquigarrow}_{S^{D_{k_{n+1}}}} T_j$. Since $child(D_{k_l}, D)$, $level(D_{k_l}) < level(D)$. Thus, by IH, $\widehat{T}_i^{D_{k_1}} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D_{k_1}}} \widehat{T}_1^{D_{k-1}}$, $\widehat{T}_1^{D_{k_2}} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D_{k_2}}} \widehat{T}_2^{D_{k_2}}$, …, $\widehat{T}_{n-1}^{D_{k_n}} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D_{k_n}}} \widehat{T}_n^{D_{k_n}}$, and $\widehat{T}_n^{D_{k_{n+1}}} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D_{k_{n+1}}}} \widehat{T}_j^{D_{k_{n+1}}}$. Hence by definition of the serialization function sf, $sf(T_i, D_{k_1}) \prec_S sf(T_1, D_{k_1})$, $sf(T_1, D_{k_2}) \prec_S sf(T_2, D_{k_2})$, …, $sf(T_{n-1}, D_{k_n}) \prec_S sf(T_n, D_{k_n})$, and $sf(T_n, D_{k_{n+1}}) \prec_S sf(T_j, D_{k_{n+1}})$. Thus, by definition of $\widehat{T}_i$, $\widehat{T}_i^D \rightsquigarrow_{\widehat{S}^D} \widehat{T}_1^D$, $\widehat{T}_1^D \rightsquigarrow_{\widehat{S}^D} \widehat{T}_2^D$, ..., $\widehat{T}_{n-1}^D \rightsquigarrow_{\widehat{S}^D} \widehat{T}_n^D$, $\widehat{T}_n^D \rightsquigarrow_{\widehat{S}^D} \widehat{T}_j^D$. Hence, $\widehat{T}_i^D \overset{*}{\rightsquigarrow}_{\widehat{S}^D} \widehat{T}_j^D$. Hence proved. $\Box$

Lemma 5: Let T_i, T_j be transactions and let D be a domain such that $\widehat{T}_i^D \overset{*}{\rightsquigarrow}_{\widehat{S}^D} \widehat{T}_j^D$. For all D', $D' \sqsubset D$, if T_i and T_j execute in D', then $sf(T_i, D') \prec_S sf(T_j, D')$.

Proof: Say there exists a D' such that $sf(T_j, D') \prec_S sf(T_i, D')$. Hence there exists a domain D'' such that $D'' \sqsubseteq D$, $parent(D'', D')$ such that $\widehat{T}_j^{D''} \rightsquigarrow_{\widehat{S}^{D''}} \widehat{T}_i^{D''}$. Hence by Lemma 3, $\widehat{T}_j^D \rightsquigarrow_{\widehat{S}^D} \widehat{T}_i^D$. Thus, $\widehat{S}^D$ is not serializable which is a contradiction. Hence, such a D' does not exist. Thus, for all D', $D' \sqsubset D$, $sf(T_i, D) \prec_S sf(T_j, D)$. $\Box$

Lemma 6: Let $T_1, T_2, \ldots, T_n$, $n > 2$, be transactions such that $T_1 \rightsquigarrow_{S_1} T_2$, $T_2 \rightsquigarrow_{S_2} T_3$, ..., $T_{n-1} \rightsquigarrow_{S_{n-1}} T_n$. Let D' and D'', $D' \in TOP$, $D'' \in TOP$, be domains such that $DB_1 \sqsubseteq D'$ and $DB_{n-1} \sqsubseteq D''$. There exists a path (D', D_1), (D_1, D_2), ..., (D_{r-1}, D_r), (D_r, D'') in LDG such that for all D_i, $i = 1, 2, \ldots, r$, there exists a DB_j, $j = 1, 2, \ldots, n-1$, such that $DB_j \sqsubset D_i$. Further, let edges (D', D_1), (D_1, D_2), ..., (D_{r-1}, D_r), (D_r, D'') have labels $L_1, L_2, \ldots, L_r$ respectively. For all L_i, $i = 1, 2, \ldots, r$, there exists a DB_j, $j = 1, 2, \ldots, n-1$, such that $DB_j \sqsubset L_i$.

Proof: The proof is by induction on n.

Basis $(n = 3)$**:** Thus, $T_1 \rightsquigarrow_{S_1} T_2$ and $T_2 \rightsquigarrow_{S_2} T_3$, where $DB_1 \sqsubset D'$ and $DB_2 \sqsubset D''$. Consider the domain $D \in TOP$ such that $Dom(T_2) \sqsubseteq D$. If $D = D'$, then since $DB_2 \sqsubset D''$ and $DB_1 \sqsubset D'$, there is an edge (D', D'') in LDG. Further, $DB_2 \sqsubset label(D', D'')$. Else, if $D = D''$, then since $DB_1 \sqsubset D'$ and $DB_2 \sqsubset D''$, there is an edge (D', D'') in LDG. Further, $DB_1 \sqsubset label(D', D'')$. Else, if $D \neq D'$ and $D \neq D''$, then there are edges (D', D) and (D, D'') in LDG such that $DB_1 \sqsubset label(D', D)$ and $DB_2 \sqsubset label(D, D'')$.

Induction: Assume that the lemma holds for $n = m-1$, $m \geq 4$. We show it holds for $n = m$. Thus, we have $T_1 \rightsquigarrow_{S_1} T_2$, $T_2 \rightsquigarrow_{S_2} T_3$, ..., $T_{m-1} \rightsquigarrow_{S_{m-1}} T_m$. Let $DB_{m-2} \sqsubset D'''$, where $D''' \in TOP$. If $D''' = D'$, then by base case, the lemma holds. Else, if $D''' = D''$, then since $T_1 \rightsquigarrow_{S_1} T_2$, $T_2 \rightsquigarrow_{S_2} T_3$, ..., $T_{m-2} \rightsquigarrow_{S_{m-2}} T_{m-1}$. Hence, by IH, the lemma holds. Else, $D''' \neq D'$ and $D''' \neq D''$. By IH, there exists a path (D', D_1), (D_1, D_2), ..., (D_{r-1}, D_r), (D_r, D''') in LDG such that for all D_i, $i = 1, 2, \ldots, r$, there exists a DB_j, $j = 1, 2, \ldots, m-2$, such that $DB_j \sqsubset D_i$. Further, for all L_i, $i = 1, 2, \ldots, r$, there exists a DB_j, $j = 1, 2, \ldots, m-2$, such that $DB_j \sqsubset L_i$. By the base case, since $T_{m-2} \rightsquigarrow_{S_{m-2}} T_{m-1}$ and $T_{m-1} \rightsquigarrow_{S_{m-1}} T_m$, there exists a path (D''', D_1'), (D_1', D_2'), ...,$(D_{r'-1}', D_{r'}')$, $(D_{r'}', D'')$ such that for all D_i', $i = 1, 2, \ldots, r'$, there exists a DB_j, $j = m-2, m-1$, such that $DB_j \sqsubset D_i'$. Further, for all L_i, $i = 1, 2, \ldots, r'$, there exists a DB_j, $j = m-2, m-1$, such that $DB_j \sqsubset L_i$. Hence, for some s there exists a path (D', D_1), (D_1, D_2), ..., (D_{s-1}, D_s), (D_s, D'') in LDG such that for all D_i, $i = 1, 2, \ldots, s$, there exists a DB_j, $j = 1, 2, \ldots, m-1$, such that $DB_j \sqsubset D_i$. Further, for all L_i, $i = 1, 2, \ldots, s$, there exists a DB_j, $j = 1, 2, \ldots, m-1$, such that $DB_j \sqsubset L_i$. $\Box$

Lemma 7: Let the set Δ satisfy restriction **R1** and the LDG be acyclic. Further, let T_i, T_j be transactions and $D, D' \in TOP$ be domains such that $global(T_i, D')$ and $global(T_j, D')$. If $\widehat{T}_i^D \overset{*}{\leadsto}_{\widehat{S}^D} \widehat{T}_j^D$, then $\widehat{T}_i^{D'} \overset{*}{\leadsto}_{\widehat{S}^{D'}} \widehat{T}_j^{D'}$.

Proof: There are two cases to consider.

- ($D \cap D' \neq \emptyset$:) We first show that both T_i and T_j execute at $D \cap D'$. If T_i does not execute at $D \cap D'$, then since T_i executes at D, there exists a $DB_1 \sqsubset D$ and a $DB_2 \sqsubset D'$ such that T_i executes at DB_1 and DB_2, where $DB_1 \not\sqsubset D \cap D'$ and $DB_2 \not\sqsubset D \cap D'$. Since T_i executes at DB_1 and DB_2, there exists a domain $D'' \in TOP$, $Dom(T_i) \sqsubset D''$, such that $D'' \neq D$ and $D'' \neq D'$. Consider the labeled domain graph LDG. In LDG since $DB_1 \sqsubset D$ and $DB_1 \sqsubset D''$, there is an edge (D, D'') such that $DB_1 \sqsubset label(D, D'')$. Further, since $DB_2 \sqsubset D'$ and $DB_2 \sqsubset D''$, there is an edge (D', D'') such that $DB_2 \sqsubset label(D, D'')$. Since $D \cap D' \neq \emptyset$, there exists an edge (D, D') in LDG. Thus, LDG contains a cycle (D, D''), (D'', D'), (D', D). Since $DB_1 \not\sqsubset D'$, $DB_1 \not\sqsubset label(D, D')$. Further, since $DB_2 \not\sqsubset D$, $DB_2 \not\sqsubset label(D, D')$. Hence, the cycle (D, D''), (D'', D'), (D', D) is a undesirable cycle. Thus, it must be the case that T_i executes in $D \cap D'$. Similarly, it is the case that T_j executes in $D \cap D'$. Since $\widehat{T}_i^D \overset{*}{\leadsto}_{\widehat{S}^D} \widehat{T}_j^D$, by Lemma 5, we have that $sf(T_i, D \cap D') \prec_S sf(T_j, D \cap D')$. Thus, by Lemma 3, since $global(T_i, D')$ and $global(T_j, D')$, we have that $\widehat{T}_i^{D'} \overset{*}{\leadsto}_{\widehat{S}^{D'}} \widehat{T}_j^{D'}$.
- ($D \cap D' = \emptyset$:) Since T_i executes at D as well as D', let T_i executes at DB_1, DB_3, where $DB_1 \sqsubset D$ and $DB_3 \sqsubset D'$. Further since T_j executes at D as well as D', let T_j executes at DB_2, DB_4, where $DB_2 \sqsubset D$ and $DB_4 \sqsubset D'$. We show that there exists a domain D'' such that $DB_1 \sqsubset D''$, $DB_2 \sqsubset D''$, $DB_3 \sqsubset D''$, $DB_4 \sqsubset D''$. Say such a domain D'' does not exist. Since T_i executes at DB_1 and DB_3, there exists a domain $D''' \in TOP$, such that $Dom(T_i) \sqsubset D'''$ and thus $DB_1 \sqsubset D'''$ and $DB_3 \sqsubset D'''$. Further, since T_j executes at DB_2 and DB_4, there exists a domain $D'''' \in TOP$, such that $Dom(T_j) \sqsubset D''''$ and thus $DB_2 \sqsubset D''''$ and $DB_4 \sqsubset D''''$. If $D''' = D''''$, then $DB_1 \sqsubset D'''$, $DB_2 \sqsubset D'''$, $DB_3 \sqsubset D'''$, and $DB_4 \sqsubset D'''$. Hence, $D''' \neq D''''$. Thus, $D \neq D' \neq D''' \neq D''''$. Consider the labeled domain graph LDG. In LDG, there is an edge (D, D''') such that $DB_1 \sqsubset label(D, D''')$, there is an edge (D''', D') such that $DB_3 \sqsubset label(D, D''')$, there is an edge (D', D'''') such that $DB_4 \sqsubset label(D', D'''')$, and there is an edge (D'''', D) such that $DB_2 \sqsubset label(D'''', D)$. Thus, LDG contains a cycle (D, D'''), (D''', D'), (D', D''''), (D'''', D). We next show that LDG contains a undesirable cycle. There are two cases to consider:
  - ($D''' \cap D'''' = \emptyset$:) Since $DB_1 \sqsubset label(D, D''')$, $DB_1 \sqsubset D'''$. Since $D''' \cap D'''' = \emptyset$, $DB_1 \not\sqsubset D''''$. Thus, $DB_1 \not\sqsubset label(D'''', D)$ and further $DB_1 \not\sqsubset label(D'''', D')$. Similarly, since $DB_1 \not\sqsubset D'$, $DB_1 \not\sqsubset label(D', D''')$. Hence $label(D, D''') \neq label(D', D''')$, $label(D, D''') \neq label(D', D'''')$, and $label(D, D''') \neq label(D, D'''')$. Using similar reasoning, we can show

that $label(D, D''') \neq label(D''', D') \neq label(D', D'''') \neq label(D'''', D)$. Hence, the cycle (D, D'''), (D''', D'), (D', D''''), (D'''', D) is a undesirable cycle.

- $(D''' \cap D'''' \neq \emptyset:)$ If $D''' \cap D'''' \neq \emptyset$, then LDG contains an edge (D'''', D''') and thus LDG besides containing the cycle (D, D'''), (D''', D'), (D', D''''), (D'''', D), also contains cycles (D, D'''), (D''', D''''), (D'''', D) and (D', D'''), (D''', D''''), (D'''', D'). Note that since $D \cap D' = \emptyset$, it must be the case that $DB_1 \not\sqsubset label(D', D''')$ and $DB_1 \not\sqsubset label(D', D'''')$. Further, $DB_2 \not\sqsubset label(D', D''')$ and $DB_2 \not\sqsubset label(D', D'''')$. Similarly, $DB_3 \not\sqsubset label(D', D''')$, $DB_3 \not\sqsubset label(D', D'''')$, $DB_4 \not\sqsubset label(D, D''')$ and $DB_4 \not\sqsubset label(D, D'''')$. Thus, if the cycle (D, D'''), (D''', D'), (D', D''''), (D'''', D) is not a undesirable cycle, then either $label(D, D'''') = label(D, D''')$ or $label(D', D'''') = label(D', D''')$. Note that $label(D, D'''') = label(D, D''')$ and $label(D', D'''') = label(D', D''')$ both cannot hold since then D_3 would be such that $DB_1 \sqsubset D_3$, $DB_2 \sqsubset D_3$, $DB_3 \sqsubset D_3$, and $DB_4 \sqsubset D_3$. If $label(D, D'''') = label(D, D''')$, and $label(D', D'''') \neq label(D', D''')$, then the cycle (D', D'''), (D''', D''''), (D'''', D') is a undesirable cycle. Else, if $label(D', D'''') = label(D', D''')$, and $label(D, D'''') \neq label(D, D''')$, then the cycle (D, D'''), (D''', D''''), (D'''', D) is a undesirable cycle.

Hence, there must exist a domain D'' such that $DB_1 \sqsubset D''$, $DB_2 \sqsubset D''$, $DB_3 \sqsubset D''$, and $DB_4 \sqsubset D''$. Since $global(T_i, D \cap D'')$ and $global(T_j, D \cap D'')$, and $\widehat{T}_i^D \overset{*}{\leadsto}_{\widehat{S}^D} \widehat{T}_j^D$, by Lemma 5, $sf(T_i, D \cap D'') \prec_S sf(T_j, D \cap D'')$. Hence by Lemma 3, and the definition of $\widehat{T}_i$, $\widehat{T}_i^{D''} \overset{*}{\leadsto}_{\widehat{S}^{D''}} \widehat{T}_j^{D''}$. Since $global(T_i, D' \cap D'')$ and $global(T_j, D' \cap D'')$, and $\widehat{T}_i^{D''} \overset{*}{\leadsto}_{\widehat{S}^{D''}} \widehat{T}_j^{D''}$, by Lemma 5, $sf(T_i, D' \cap D'') \prec_S sf(T_j, D' \cap D'')$. Hence by Lemma 3, and the definition of $\widehat{T}_i$, $\widehat{T}_i^{D'} \overset{*}{\leadsto}_{\widehat{S}^{D'}} \widehat{T}_j^{D'}$. Hence proved. $\square$

Lemma 8: Let the set Δ of domains satisfy the restriction **R1** and let LDG be acyclic. Further, let D be a domain in Δ, $level(D) \geq 2$, and T_1 and T_n be transactions such that $global(T_1, D)$ and $global(T_n, D)$. If $T_1 \leadsto_{DB_1} T_2$[5], $T_2 \leadsto_{DB_2} T_3$, ..., $T_{n-1} \leadsto_{DB_{n-1}} T_n$, then $\widehat{T}_1^D \overset{*}{\leadsto}_{\widehat{S}^D} \widehat{T}_n^D$.

Proof: The proof in by induction on n.

Basis ($n = 1$): Thus, $T_1 \leadsto_{DB_1} T_2$. There are two cases to consider.

- $(DB_1 \sqsubseteq D:)$ In this case, $T_1 \leadsto_{S^D} T_2$. Thus, by Lemma 4, since $global(T_1, D)$ and $global(T_2, D)$, we have that $\widehat{T}_1^D \overset{*}{\leadsto}_{\widehat{S}^D} \widehat{T}_2^D$.
- $(DB_1 \not\sqsubseteq D:)$ Let $DB_1 \sqsubset D'$, where $D' \in TOP$. By Lemma 4, $\widehat{T}_1^{D'} \overset{*}{\leadsto}_{\widehat{S}^{D'}} \widehat{T}_2^{D'}$. Since $global(T_1, D)$ and $global(T_2, D)$, by Lemma 7, $\widehat{T}_1^D \overset{*}{\leadsto}_{\widehat{S}^D} \widehat{T}_n^D$.

Induction: Assume that the lemma holds for all $n \leq m - 1$. We show that it holds for $n = m$. Thus, we have $T_1 \leadsto_{DB_1} T_2$, $T_2 \leadsto_{DB_2} T_3$, ..., $T_{m-1} \leadsto_{DB_{m-1}} T_m$. There are two cases to consider.

- (there exists i, $i = 1, 2, 3, \ldots, m-1$, $DB_i \sqsubset D$:) Let $DB_k \sqsubset D$, $1 \leq k \leq m-1$. Further, let DB_{k_1} and DB_{k_2}, $1 \leq k_1 \leq k$, and $k \leq k_2 \leq m-1$, be such that for all DB_i, $i = k_1, k_1+1, \ldots, k, k+1, \ldots, k_2$, $DB_i \sqsubset D$, $DB_{k_1-1} \not\sqsubset D$ and $DB_{k_2+1} \not\sqsubset D$. If $k_1 = 1$ and

[5] For notational brevity, we denote $\leadsto_{S^{DB_i}}$ by $\leadsto_{DB_i}$.

$k_2 = m-1$, then by Lemma 4, since for all DB_i, $i = 1,2,\ldots,m-1$, $DB_i \sqsubset D$, $\widehat{T}_1^D \overset{*}{\rightsquigarrow}_{\widehat{S}^D} \widehat{T}_m^D$. So we only need to consider the case in which either $1 < k_1$ or $k_2 < m-1$. There are two cases to consider:

- ($1 < k_1$:) Consider transaction T_{k_1}. Note that $T_{k_1-1} \rightsquigarrow_{DB_{k_1-1}} T_{k_1}$ and further $T_{k_1} \rightsquigarrow_{DB_{k_1}} T_{k_1+1}$. Since $DB_{k_1-1} \not\sqsubseteq D$, $DB_{k_1} \sqsubseteq D$, and transaction T_{k_1} executes on DB_{k_1} and DB_{k_1-1}, $global(T_{k_1}, D)$. Hence, by IH, $\widehat{T}_1^D \overset{*}{\rightsquigarrow}_{\widehat{S}^D} \widehat{T}_{k_1}^D$ and further $\widehat{T}_{k_1}^D \overset{*}{\rightsquigarrow}_{\widehat{S}^D} \widehat{T}_n^D$. Hence, $\widehat{T}_1^D \overset{*}{\rightsquigarrow}_{\widehat{S}^D} \widehat{T}_n^D$.
- ($k_2 < m$:) Consider transaction T_{k_2}. Note that $T_{k_2-1} \rightsquigarrow_{DB_{k_2-1}} T_{k_2}$ and further $T_{k_2} \rightsquigarrow_{DB_{k_2}} T_{k_2+1}$. Since $DB_{k_2+1} \not\sqsubseteq D$, $DB_{k_2} \sqsubseteq D$, and transaction T_{k_2} executes on DB_{k_2} and DB_{k_1+1}, $global(T_{k_2}, D)$. Hence, by IH, $\widehat{T}_1^D \overset{*}{\rightsquigarrow}_{\widehat{S}^D} \widehat{T}_{k_2}^D$ and further $\widehat{T}_{k_2}^D \overset{*}{\rightsquigarrow}_{\widehat{S}^D} \widehat{T}_n^D$. Hence, $\widehat{T}_1^D \overset{*}{\rightsquigarrow}_{\widehat{S}^D} \widehat{T}_n^D$.

- (for all i, $i = 1,2,3,\ldots,m-1$, $DB_i \not\sqsubset D$:) Let $DB_1 \sqsubset D'$, $D' \neq D$, where $D' \in TOP$ and $Dom(T_1) \sqsubseteq D'$. There are two cases to consider:
  - ($DB_{m-1} \sqsubseteq D'$:) We first show that $\widehat{T}_1^{D'} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D'}} \widehat{T}_m^{D'}$. It will follow from Lemma 7 that $\widehat{T}_1^D \overset{*}{\rightsquigarrow}_{\widehat{S}^D} \widehat{T}_m^D$. Since $DB_{m-1} \sqsubseteq D'$, we have that $T_1 \rightsquigarrow_{DB_1} T_2$, $T_2 \rightsquigarrow_{DB_2} T_3$, …, $T_{m-1} \rightsquigarrow_{DB_{m-1}} T_m$, where $DB_1, DB_{m-1} \sqsubset D'$. If for all DB_i, $i = 1,2,\ldots,m-1$, $DB_i \sqsubset D'$, then since $T_1 \overset{*}{\rightsquigarrow}_{S^{D'}} T_m$, by Lemma 4, we have that $\widehat{T}_1^{D'} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D'}} \widehat{T}_m^{D'}$. Thus, by Lemma 7, $\widehat{T}_1^D \overset{*}{\rightsquigarrow}_{\widehat{S}^D} \widehat{T}_m^D$. Else, there exists a DB_k, $k = 2,3,\ldots,m-2$, such that $DB_k \not\sqsubset D'$. Let k_1 be such that $DB_{k_1} \not\sqsubset D'$ and for all $k = 1,2,\ldots,k_1-1$, $DB_k \sqsubset D'$. Thus, $T_{k_1-1} \rightsquigarrow_{DB_{k_1-1}} T_{k_1}$ and $T_{k_1} \rightsquigarrow_{DB_{k_1}} T_{k_1+1}$, where $DB_{k_1-1} \sqsubset D'$ and $DB_{k_1} \not\sqsubset D'$. Since T_{k_1} executes both on DB_{k_1-1} and DB_{k_1}, $global(T_{k_1}, D')$. Hence by IH, $\widehat{T}_1^{D'} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D'}} \widehat{T}_{k_1}^{D'}$ and $\widehat{T}_{k_1}^{D'} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D'}} \widehat{T}_m^{D'}$. Hence, $\widehat{T}_1^{D'} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D'}} \widehat{T}_m^{D'}$. Thus, by Lemma 7, $\widehat{T}_1^D \overset{*}{\rightsquigarrow}_{\widehat{S}^D} \widehat{T}_m^D$.
  - ($DB_{m-1} \not\sqsubseteq D'$:) Let $DB_{m-1} \sqsubset D''$, where $D'' \in TOP$ and $Dom(T_m) \sqsubseteq D''$. Note that $D'' \neq D'$ and further $D'' \neq D$. Since T_1 executes in both D and D', and $Dom(T_1) \sqsubseteq D'$, LDG contains an edge (D, D'). Let $label(D, D') = L'$. Similarly, since T_m executes in both D and D'', and $Dom(T_m) \sqsubseteq D''$, LDG contains an edge (D, D''). Let $label(D, D'') = L''$. We first show that it must be the case that $L' = L''$.

    Assume on the contrary that $L' \neq L''$. Since $T_1 \rightsquigarrow_{DB_1} T_2, T_2 \rightsquigarrow_{DB_2} T_3, \ldots, T_{m-1} \rightsquigarrow_{DB_{m-1}} T_m$, where $DB_1 \sqsubset D'$ and $DB_{m-1} \sqsubset D''$, by Lemma 6, there exists a path (D', D_1), (D_1, D_2), …, (D_{r-1}, D_r), (D_r, D'') such that for all D_i, $i = 1,2,\ldots,r$, there exists a DB_j, $j = 1,2,\ldots,m-1$, $DB_j \sqsubset D_i$ and further, for all edges in the path (D_l, D_m), there exists a DB_j, $j = 1,2,\ldots,m-1$, $DB_j \sqsubset label(D_l, D_m)$. Since for all DB_j, $DB_j \not\sqsubset D$, the path does not contain D. Hence, LDG contains a cycle (D, D'), (D', D_1), (D_1, D_2), …, (D_{r-1}, D_r), (D_r, D''), (D'', D). We next show, using induction on r, that LDG contains a undesirable cycle.

    Basis ($r = 0$:) Thus, LDG contains an edge (D', D''). Let $label(D', D''') = L'''$. Since there exists a DB_j such that $DB_j \sqsubset L'''$, and further since $L' \sqsubset D$ and $L'' \sqsubset D$, it is the case that $L''' \neq L''$ and $L''' \neq L''$. Since by assumption $L' \neq L''$, the cycle, (D, D'), (D', D''), (D'', D) is a undesirable cycle.

Induction: Assume that if there is a cycle (D, D'), (D', D_1), (D_1, D_2), …, (D_{r-2}, D_{r-1}), (D_{r-1}, D''), (D'', D), then LDG contains a undesirable cycle. We next show that if there exists a cycle (D, D'), (D', D_1), (D_1, D_2), …, (D_{r-1}, D_r), (D_r, D''), (D'', D) in the LDG, then LDG contains a undesirable cycle. Consider the cycle (D, D'), (D', D_1), (D_1, D_2), …, (D_{r-1}, D_r), (D_r, D''), (D'', D). Let the labels on the edges (D', D_1), (D_1, D_2), …, (D_{r-1}, D_r), (D_r, D'') be $L_1, L_2, \ldots, L_r$ respectively. By assumption $L' \neq L''$. Further, since for each L_i, there exists a DB_j, $j = 1, 2, \ldots, m-1$ such that $DB_j \sqsubset L_i$, and since $L' \sqsubset D$, $L'' \sqsubset D$, and since for all DB_j, $j = 1, 2, \ldots, m-1$, $DB_j \not\sqsubset D$, it is the case that $L' \neq L_i$ and $L'' \neq L_i$, for all $i = 1, 2, \ldots, r$. Thus, if the cycle (D, D'), (D', D_1), (D_1, D_2), …, (D_{r-2}, D_{r-1}), (D_{r-1}, D''), (D'', D), is not a undesirable cycle, then there must exist labels L_{r_1}, L_{r_2} in the cycle such that $L_{r_1} = L_{r_2}$. Consider domains D_{r_1-1}, D_{r_2+1}. Since $L_{r_1} \sqsubset D_{r_1-1}$ and $L_{r_2} \sqsubset D_{r_2+1}$, there is an edge between D_{r_1-1} and D_{r_2+1} with a label L such that $L_{r_1} \sqsubset L$. Hence, there exists a cycle in LDG, (D, D'), (D', D_1), (D_1, D_2), …, (D_{r_1-1}, D_{r_2+1}), (D_{r_2+1}, D_{r_2+2}), …, (D_{r-1}, D_r), (D_r, D''), (D'', D) such that the length of the cycle is less than r. Thus, by IH, there exists a undesirable cycle in LDG.

Hence it must be the case that $L' = L''$. Since $Dom(T_m) \sqsubseteq D''$, and T_m executes in D, T_m executes in L''. Since $L'' = L'$ and $L' \sqsubset D'$, it is the case that T_m executes in D'. Further since $global(T_m, D)$, it is the case that $global(T_m, D')$. Hence, we have that $T_1 \rightsquigarrow_{DB_1} T_2$, $T_2 \rightsquigarrow_{DB_2} T_3$, …, $T_{m-1} \rightsquigarrow_{DB_{m-1}} T_m$, where $global(T_1, D')$, $global(T_m, D')$ and $DB_1 \sqsubset D'$ and $DB_{m-1} \sqsubset D''$. Since $D' \neq D''$, there exists a k such that for all i, $1 \leq i < k$, $DB_k \sqsubset D'$ and $DB_k \not\sqsubset D'$. Hence, since $T_{k-1} \rightsquigarrow_{DB_{k-1}} T_k$, and $T_k \rightsquigarrow_{DB_k} T_{k+1}$, where $DB_{k-1} \sqsubset D'$ and $DB_k \not\sqsubset D'$, we have that $global(T_k, D')$. Thus, by IH, we have that $\widehat{T}_1^{D'} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D'}} \widehat{T}_k^{D'}$ and $\widehat{T}_k^{D'} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D'}} \widehat{T}_n^{D'}$. Thus, $\widehat{T}_1^{D'} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D'}} \widehat{T}_n^{D'}$. Hence, by Lemma 7, $\widehat{T}_1^{D} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D}} \widehat{T}_n^{D}$. Hence proved. □

Proof of Theorem 4 (cont.): If S is not serializable, then there exists transactions $T_1, T_2, \ldots, T_n$ such that $T_1 \rightsquigarrow_{DB_1} T_2$, $T_2 \rightsquigarrow_{DB_2} T_3$, …, $T_{n-1} \rightsquigarrow_{DB_{n-1}} T_n$, $T_n \rightsquigarrow_{DB_n} T_1$. Let $D \in TOP$ such that $DB_1 \sqsubset D$. If for all DB_i, $i = 1, 2, \ldots, n$, $DB_i \sqsubset D$, then $T_1 \overset{*}{\rightsquigarrow}_{S^D} T_1$. Hence, by Lemma 4, $\widehat{T}_1^{D} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D}} \widehat{T}_1^{D}$ which is a contradiction. Thus, there exists a $DB_k \not\sqsubset D$. Let for all DB_i, $1 \leq i < k$, $DB_i \sqsubset D$ and further $DB_k \not\sqsubset D$. Hence, since transaction T_k executes on both DB_{k-1} and DB_k, $global(T_k, D)$. Consider the sequence of conflicts $T_k \rightsquigarrow_{DB_k} T_{k+1}$, $T_{k+1} \rightsquigarrow_{DB_{k+1}} T_{k+2}$, …, $T_{n-1} \rightsquigarrow_{DB_{n-1}} T_n$, $T_n \rightsquigarrow_{DB_n} T_1$, $T_1 \rightsquigarrow_{DB_1} T_2$, …, $T_{k-1} \rightsquigarrow_{DB_{k-1}} T_k$. Since $global(T_k, D)$, by Lemma 8, $\widehat{T}_k^{D} \overset{*}{\rightsquigarrow}_{\widehat{S}^{D}} \widehat{T}_k^{D}$ which is a contradiction. Hence, the sequence of transactions cannot exist. Thus, S is serializable. □