transaction may not precede all other operations of the subtransaction) in the L_r model. If there are no integrity constraints between local and global data items, and global transactions have fixed-structure, then S is strongly correct.

Proof We present the outline of the proof. In the L_r model, since global transactions only read and write global data items, only global transactions write global data items, by Corollary 3, the global conjunct is preserved. Further, local transactions can be shown to read consistent global data items using Lemma 15 (since S can easily be modified to a schedule containing a single local transaction). Local conjuncts can be shown to be preserved using Corollary 4. Hence proved. □

Corollary 7 *Let S be a G2LPL schedule (in which the write on $ticket_i$ by a global subtransaction may not precede all other operations of the subtransaction) in the $G_{rw}L_r$ model. If there are no integrity constraints between local and global data items, and both local and global transactions have fixed-structure, then S is strongly correct.*

Proof By induction, it can be shown that for an arbitrary operation $p \in S$,

1. for all $lt \in \tau_L$, if $brelease(lt, p)$ and lt executes at a 2PL site, then $read(lt^{GD})$ is consistent.
2. for all $lt \in \tau_L$, if lt executes at a non-2PL site, then $read(before(lt^{GD}, p, S))$ is consistent.
3. for all $gt \in \tau_G$, $read(before(gt, p, S))$ is consistent.

Using Lemma 15, 1 and 2 can be shown to be true. The proof of 3 is similar to Lemma 19 (cases 1 and 2). $read(before(gt, p, S))$ can be trivially shown to be consistent using Lemma 13 in case $p \in LD_i$, for some i, $i = m+1, m+2, \ldots, m+q$ (since both local and global transactions have fixed-structure). □

Proof: Let DS_1 be a consistent database state such that $legal(DS_1, S)$. Let $\{DS_1\}S\{DS_2\}$. In order to show that S is strongly correct, we need to show DS_2 to be consistent. Since no integrity constraints are present between local and global data items, the integrity constraints can be viewed as $IC = L_1 \wedge L_2 \wedge \ldots \wedge L_m \wedge L_{m+1} \wedge L_{m+2} \wedge L_{m+q} \wedge G$, where L_i is a conjunct defined over data items in LD_i and G is a conjunct defined over data items in GD. As a result of the model $LD_i \cap LD_j = \emptyset$ and $LD_i \cap GD = \emptyset$.

We now show that G is preserved by S. Since only global transactions write on global data items, $\tau_w(GD, S) \subseteq \tau_G$. Since S^{τ_G} is serializable, $(S^{\tau_G})^{GD}$ is serializable. Choosing p to be the last operation in S, by Lemma 20, $read(gt)$ is consistent, for all global transactions gt. As $read(t)$ is consistent, $read(t^{D-GD})$ is also consistent, for all $t \in \tau_w(GD, S)$ By Lemma 3, since DS_1 is consistent, DS_1^{GD} is consistent. Hence, by Corollary 4, since no integrity constraints are present between local and global data items, DS_2^{GD} is consistent.

We now show that both global and local transaction executions preserve L_i, for all i, $i = 1, 2, \ldots, m+q$. Since both local and global transactions write on local data items, $\tau_w(LD_i, S) \subseteq \tau$. Since $S^\tau = S$ and S^{D_i} is serializable, $(S^\tau)^{LD_i}$ is serializable. Let $t_1, t_2, \ldots, t_n$ be a serialization order of transactions on $(S^\tau)^{LD_i}$. Choosing p to be the last operation in S, by Lemma 20, $read(lt^{GD_i})$ is consistent, for all local transactions lt. As a result, if $t_j \in \tau_G$, then since $read(t_j)$ is consistent, $read(t_j^{D-LD_i})$ is also consistent. If, on the other hand, $t_j \in \tau_L$, then since local transactions executing at site s_i access only data items at site s_i and $read(t_j^{GD_i})$ is consistent, $read(t_j^{D-LD_i})$ is also consistent. Thus, $read(t_j^{D-LD_i})$, for all $j = 1, 2, \ldots, n$ is consistent. Since DS_1 is consistent, by Lemma 3, $DS_1^{LD_i}$ is consistent. Thus, by Corollary 4, since no integrity constraints are present between local and global data items, $DS_2^{LD_i}$ is consistent and $state(t, LD_i, S, DS_1)$, $t \in \tau_L$ is consistent. Further, since t^{GD_i} is consistent, for all $t \in \tau_L$, no integrity constraints are present between local and global data items, and local transactions executing at site s_i only access data items at site s_i, $read(t)$ is consistent, for all $t \in \tau_L$.

Thus DS_2^{GD} and $DS_2^{LD_i}$, for all i, $i = 1, 2, \ldots, m+q$, are consistent. Hence, by Lemma 3, DS_2 is consistent. Thus, S is strongly correct. □

Corollary 6 *Let S be a G2LPL schedule (in which the write on $ticket_i$ by a global sub-*

Case 2: t_k is a local transaction executing at site s_i. Since, $read(t_k) \subseteq state(t_k, D_i, S, DS_1)$, and by IH, $state(t_k, D_i, S, DS)$ is consistent, by Lemma 4, $state(t_{k+1}, D_i, S, DS)$ is consistent.

Case 3: t_k is a global transaction. As mentioned earlier, t_k cannot execute at site s_i. Thus, $state(t_{k+1}, D_i, S, DS) = state(t_k, D_i, S, DS)$, which is consistent.

By a similar argument we can prove that $state(gt, D_i, S, DS)$ is consistent. Thus, since $read(before(gt^{LD_i}, p, S)) \subseteq state(gt, D_i, S, DS)$, and no integrity constraints are present between local and global data items, by IH, $Lread(before(gt^{D1}, p, S)) \cup read(before(gt^{D2}, p, S))$ is consistent. □

Lemma 20 *Let S be a G2LPL schedule in the $G_{rw}L_r$ model, and p be an arbitrary operation in S. If there are no integrity constraints between local and global data items, and global transaction programs have fixed-structure, then*

- *for all $lt \in \tau_L$, if $brelease(lt, p)$ and lt executes at a 2PL site, then $read(lt^{GD})$ is consistent.*
- *for all $lt \in \tau_L$, if lt executes at a non-2PL site, then $read(before(lt^{GD}, p, S))$ is consistent.*
- *for all $gt \in \tau_G$, $Lread(before(gt^{D1}, p, S)) \cup read(before(gt^{D2}, p, S))$ is consistent.*

Proof: Let DS be a consistent database state such that $legal(DS, S)$. The proof is by induction on $depth(p)$.
Basis ($depth(p) = 0$): Trivial, Since $DS^{d'}$ is consistent, for all $d' \in D$.
Induction: Assume true for $depth(p) = k$. We need to show the result holds for $depth(p) = k+1$. Let $p \in lt$. If lt executes at a 2PL site and $brelease(lt, p)$ is not true, then by IH, the lemma holds. If lt executes at a non-2PL site or if lt executes at a 2PL site and $brelease(lt, p)$ is true then by IH, and Lemma 17, the lemma is true. If $p \in gt$, then by Lemma 19 and IH, $Lread(before(gt^{D1}, p, S)) \cup read(before(gt^{D2}, p, S))$ is consistent. Hence proved. □

Theorem 10 *Let S be a G2LPL schedule in the $G_{rw}L_r$ model. If there are no integrity constraints between local and global data items, and global transactions have fixed-structure, then S is strongly correct.*

$Lread(before(gt^{LD_i}, p, S)) \subseteq state(gt, LD_i - \bigcup_{k=1}^{r} WS(after(t_k, p, S)) - \bigcup_{t \in C(gt, LD_i, p, S)} WS(t), DS, S$

Since $Lread(before(gt^{D1}, p', S)) \cup read(before(gt^{D2}, p, S))$ is consistent, and no integrity constraints are present between local and global data items, $Lread(before(gt^{D1}, p, S)) \cup read(gt^{D2}, p, S))$ is consistent.

Case 3: $entity(p) \in LD_i$, for some i, $i = m+1, m+2, \ldots, m+q$ (s_i is a non-2PL site).

Since no integrity constraints are present between local and global data items, the integrity constraint can be viewed as $IC = C' \wedge C''$, where C' and C'' are defined over data items in $D' = D2 \cup \bigcup_{i=1}^{m} GD_i$, and $D'' = \bigcup_{i=1}^{m} LD_i$ respectively, and $D' \cap D'' = \emptyset$. Let $\tau' = \tau_w(D', S)$ (thus, τ' does not contain any local transaction which executes a 2PL site). Since global transactions are serialized in the same order at the non-2PL sites due to forced conflicts and S^{τ_G} is serializable, $S^{\tau'}$, and thus, $(S^{\tau'})^{D'}$ is serializable. Let a serialization order of transactions in $(S^{\tau'})^{D'}$ be $t_1, t_2, \ldots, t_r, t_{r+1}, t_{r+2}, \ldots, t_{r+s}, gt, t_{r+s+1}, t_{r+s+2}, \ldots, t_n$, such that t_r is the last global transaction that executes at site s_i and is serialized before gt in $(S^{\tau'})^{D'}$. Since S is a G2LPL schedule, t_r must write on $ticket_i$ before gt writes on $ticket_i$. Further, since gt's write operation on $ticket_i$ precedes p, t_r releases a global lock before p. Since t_r releases a global lock before p, every $t' \in (\{t_1, t_2, \ldots, t_r\} \cap \tau_G)$ also releases a global lock before p. As a result, since S is a G2LPL schedule, all the global transactions in $\{t_1, t_2, \ldots, t_r\}$ must have obtained their local locks at the 2PL sites before p. Thus, by the hypothesis of the lemma, $read(t_j^{D''}) \subseteq Lread(before(t_j^{D_1}, p, S))$, is consistent, for all j, $j = 1, 2, \ldots, r$, $t_j \in \tau_G$. For all $t_j \in \tau_L$, $j = 1, 2, \ldots, r$, $read(t_j^{D''})$ is trivially consistent, since t_j does not access data items at any of the 2PL sites. Hence, by Corollary 4, since $read(t_j^{D''})$ is consistent, for all j, $j = 1, 2, \ldots, r$, and $D' \cap D'' = \emptyset$, $state(t_{r+1}, D', S, DS)$ is consistent. As $D_i \subseteq D'$, $state(t_{r+1}, D_i, S, DS)$ is consistent. Since t_r is the last global transaction that executes at site s_i, transactions $t_{r+1}, t_{r+2}, \ldots, t_{r+s}$ are either local transactions or global transactions that do not execute at site s_i. We claim that for all j, $j = r+1, \ldots, r+s$, $state(t_j, D_i, S, DS)$ is consistent. The proof is by induction. The basis ($j = r+1$), as shown earlier is true. Assume the claim is true for $j = k$, for some k, $k = r+1, r+2, \ldots, r+s-1$. In order to show that $state(t_{k+1}, D_i, S, DS)$ is consistent, we need to consider the following cases.

Case 1: t_k is a local transaction that executes at a non-2PL site different from s_i. In this case, t_k does not access data items in D_i and thus, $state(t_{k+1}, D_i, S, DS)$ is consistent.

Lemma 19 *Let S be a G2LPL schedule in the $G_{rw}L_r$ model. Let no integrity constraints be present between local and global data items, and global transactions have fixed-structure. Let p be an arbitrary operation belonging to gt, where $gt \in \tau_G$. For every operation $p' \in S$, such that $p' \prec_S p$, if*

- *for all $lt \in \tau_L$, if $brelease(lt, p')$ and lt executes at a 2PL site, then $read(lt^{GD})$ is consistent.*
- *for all $lt \in \tau_L$, if lt executes at a non-2PL site, then $read(before(lt^{GD}, p', S))$ is consistent.*
- *for all $gt' \in \tau_G$, $Lread(before(gt'^{D1}, p', S)) \cup read(before(gt'^{D2}, p', S))$ is consistent.*

then, $Lread(before(gt^{D1}, p, S)) \cup read(before(gt^{D2}, p, S))$ is consistent.

Proof: Let DS be a consistent database state such that $legal(DS, S)$. Since global transactions access both local and global data items, we need to consider the following cases.

Case 1: $entity(p) \in GD$.
Since only global transactions write global data items, $\tau_w(d, S) \subseteq \tau_G$. Since S^{τ_G} is serializable, $(S^{\tau_G})^{GD}$ is serializable. Let $gt_1, \ldots, gt_r, gt, gt_{r+1}, \ldots, gt_n$ be a serialization order of global transactions in $(S^{\tau_G})^{GD}$. By Lemma 13, since global transactions have fixed-structure and $read(before(gt', p', S))$ is consistent, for all $gt' \in \tau_G$ and $p' \prec_S p$, $state(gt, GD - \bigcup_{j=1}^{r} WS(after(gt^{GD},$ is consistent. Using lemmas 8 and 9, it can be shown that, $(LRS(before(gt^{GD}, p, S)) \cup RS(before(gt^{GD}$ $GD - \bigcup_{j=1}^{r} WS(after(gt^{GD}, p, S))$. Thus, $Lread(before(gt^{GD}, p, S)) \cup read(before(gt^{GD}, p, S))$ is consistent. Since no integrity constraints are present between local and global data items, by IH, $Lread(before(gt^{D1}, p, S)) \cup read(before(gt^{D2}, p, S))$ is consistent. Hence proved.

Case 2: $entity(p) \in LD_i$, for some i, $i = 1, 2, \ldots, m$ (s_i is a 2PL site).
Since s_i is a 2PL site, S^{D_i} is serializable. Let $t_1, t_2, \ldots, t_r, gt, t_{r+1}, \ldots, t_n$ be a serialization order of transactions in S^{LD_i}. Since $p \in gt$, from the statement of the lemma we can conclude that for all k, $k = 1, 2, \ldots, r$, such that $t_k \in \tau_G$, $read(before(t_k, p, S)) \subseteq (Lread(before(gt^{D1}, p, S)) \cup read(before(gt^{D2}, p, S)))$ is consistent. Also, it is given that for all k, $k = 1, 2, \ldots, r$, such that $t_k \in \tau_L$ and $brelease(t_k, p)$, $read(t_k^{GD_i})$ is consistent. Thus, by Lemma 18, $state(gt, LD_i - \bigcup_{k=1}^{r} WS(after(t_k, p, S)) - \bigcup_{t \in C(gt, LD_i, p, S)} WS(t), DS, S)$ is consistent. By Lemma 10(b), $LRS(before(gt^{LD_i}, p, S)) \subseteq LD_i - \bigcup_{k=1}^{r} WS(after(t_k, p, S)) - \bigcup_{t \in C(gt, LD_i, p, S)} WS(t)$ and thus,

local data items, t_k could either be a local transaction or a global transaction. We consider the following cases:

Case 1: t_k releases its first lock on a data item in LD_i after p (t_k could be either a local or a global transaction). Trivially, since $C(t_{k+1}, LD_i, p, S) = C(t_k, LD_i, p, S) \cup t_k$, and by IH, $state(t_k, LD_i - \bigcup_{l=1}^{k-1} WS(after(t_l, p, S)) - \bigcup_{t \in C(t_k, LD_i, p, S)} WS(t), DS, S)$ is consistent, $state(t_{k+1}, LD_i - \bigcup_{l=1}^{k} WS(after(t_l, p, S)) - \bigcup_{t \in C(t_{k+1}, LD_i, p, S)} WS(t), DS, S)$ is consistent.

Case 2: $t_k \in \tau_G$ and t_k releases its first lock on a data item in LD_i before p. By IH, $state(t_k, LD_i - \bigcup_{l=1}^{k-1} WS(after(t_l, p, S)) - \bigcup_{t \in C(t_k, LD_i, p, S)} WS(t), DS, S)$ and $read(before(t_k, p, S))$ are consistent. Also, since transactions serialized before t_k release their first lock on a data item in LD_i before p, $C(t_k, LD_i, p, S) = \emptyset$. By Corollary 5(b), $read(t_k^{LD_i}) \subseteq state(t_k, LD_i - \bigcup_{l=1}^{k-1} WS(after(t_l, p, S)), DS, S)$. Since no integrity constraints are present between local and global data items, and $C(t_k, LD_i, p, S) = \emptyset$, by IH,

$$state(t_k, LD_i - \bigcup_{l=1}^{k-1} WS(after(t_l, p, S)), DS, S) \cup read(before(t_k, p, S))$$

is consistent. Thus, since global transactions have fixed-structure, by Lemma 12, $state(t_{k+1}, LD_i - \bigcup_{l=1}^{k} WS(after(t_l, p, S)), DS, S)$ is consistent (note that $C(t_{k+1}, LD_i, p, S) = \emptyset$, since $brelease(t_k, p, S)$ is true).

Case 3: $t_k \in \tau_L$ and t_k releases its first lock on a data item in LD_i before p. By IH, $state(t_k, LD_i - \bigcup_{l=1}^{k-1} WS(after(t_l, p, S)) - \bigcup_{t \in C(t_k, LD_i, p, S)} WS(t), DS, S)$ and $read(t_k^{GD})$ are consistent. Also, since transactions serialized before t_k release their first lock on a data item in LD_i before p, $C(t_k, LD_i, p, S) = \emptyset$. By Corollary 5(b), $read(t_k^{LD_i}) \subseteq state(t_k, LD_i - \bigcup_{l=1}^{k-1} WS(after(t_l, p, S)), DS, S)$. Since lt only accesses data items belonging to site s_i, and no integrity constraints are present between local and global data items, by IH, $state(t_k, LD_i - \bigcup_{l=1}^{k-1} WS(after(t_l, p, S)), DS, S) \cup read(t_k)$ is consistent. Thus, by Lemma 4, $state(t_{k+1}, LD_i - \bigcup_{l=1}^{k-1} WS(after(t_l, p, S)), DS, S)$ and thus, $state(t_{k+1}, LD_i - \bigcup_{l=1}^{k} WS(after(t_l, p, S)), DS, S)$ is consistent. (note that $C(t_{k+1}, LD_i, p, S) = \emptyset$, since $brelease(t_k, p, S)$ is true).

Thus, $state(t_j, LD_i - \bigcup_{l=1}^{j-1} WS(after(t_l, p, S)) - \bigcup_{t \in C(t_j, LD_i, p, S)} WS(t), DS, S)$ is consistent, for all j, $j = 1, 2, \ldots, n$. $\square$

S and S' execute from DS and global transactions in τ'_G and τ_G read the same local data items before p (even though, as a result of the construction of the gtp''s, gt'_j does not explicitly read local data items), $read(before(gt_j, p, S)) = read(before(gt'_j, p', S'))$, for all j, $j = 1, 2, \ldots, r$. Thus, by Lemma 14,

$state(lt, GD_i - \bigcup_{k=1}^{r} WS(after(gt_k, p, S)), S, DS) =$

$$state(lt', GD_i - \bigcup_{k=1}^{r} WS(after(gt'_k, p', S')), S', DS).$$

(Note that Lemma 14 requires t_i, t'_i to be transactions that result from the same transaction program tp_i, while above, transactions gt_i, gt'_i result from transaction programs gtp_i, gtp'_i. This does not create problems, since gtp'_i is a modification of gtp_i and the writes done by both programs are identical functions of the reads done by the programs).

By Lemma 15, since S' is a G2LPL schedule containing a single local transaction, $state(lt', D_i - \bigcup_{k=1}^{r} WS(after(gt'_k, p', S')), S', DS)$ is consistent. If s_i is a 2PL site, then by Corollary 5(b), $read(lt^{GD_i}) \subseteq state(lt, GD_i - \bigcup_{k=1}^{r} WS(after(gt_k, p, S)), S, DS)$. Hence, $read(lt^{GD_i})$ is consistent. If s_i is a non-2PL site, then by Lemma 8, $read(before(lt^{GD_i}, p, S)) \subseteq state(lt, GD_i - \bigcup_{k=1}^{r} WS(after(gt_k, p, S)), S, DS)$ and thus, $read(before(lt^{GD_i}, p, S))$ is consistent. □

In the following lemmas, we show that global transaction reads are consistent.

Lemma 18 *Let S be a G2LPL schedule in the $G_{rw}L_r$ model. Let no integrity constraints be present between local and global data items, and global transactions have fixed-structure. Let DS be a consistent database state such that $legal(DS, S)$ and p be an arbitrary operation in S. Let site s_i be a 2PL site and $t_1, t_2, \ldots, t_n$ be a serialization order of transactions in S^{LD_i} such that $t_r^{LD_i}$ releases its first lock before $t_{r+1}^{LD_i}$, for all r, $r = 1, 2, \ldots, n-1$. If, for all l, $l = 1, 2, \ldots, j-1$,*

- *if $t_l \in \tau_L$ and $brelease(t_l^{LD_i}, p, S)$, then $read(t_l^{GD})$ is consistent, and*
- *if $t_l \in \tau_G$ and $brelease(t_l^{LD_i}, p, S)$, then $read(before(t_l, p, S))$ is consistent,*

then $state(t_j, LD_i - \bigcup_{l=1}^{j-1} WS(after(t_l, p, S)) - \bigcup_{t \in C(t_j, LD_i, p, S)} WS(t), DS, S)$, for all j, $j = 1, 2, \ldots, n$, is consistent.

Proof: The proof is by induction on j.

Basis ($j = 1$): $state(t_1, LD_i, DS, S) = DS^{LD_i}$, which is consistent.

Induction: Assume that the lemma is true for $j = k$. We need to show that the lemma is true for $j = k + 1$. Since in the $G_{rw}L_r$ model both local and global transactions are permitted to write

is given that $read(before(gt^{LD}, p, S))$ is consistent. Thus, there exists a consistent database state DS_1 such that $DS_1^{RS(before(gt^{LD}, p, S))} = read(before(gt^{LD}, p, S))$. Construct global transaction program gtp' from gtp as follows : $gtp' = \{gtp\}^{ds}$, where $ds = DS_1^{LD}$. By Lemma 16, since there are no integrity constraints between local and global data items, and gtp preserves database consistency, gtp' also preserves database consistency. Note that gtp' only reads and writes global data items and has fixed-structure. Further, if gt' is the transaction that results from the execution of transaction program gtp', then $struct(gt^{GD}) = struct(gt')$. We denote the set of modified global transaction programs by $\tau p'_G$, and the transactions resulting from their execution in S' by τ'_G.

Once global transactions are modified as mentioned above, the values written by global transactions no longer depend on writes done by other local transaction. Thus, deleting local transactions have no affect on lt's reads of global data items. We now construct S' from S. Let DS be a consistent database state such that $legal(DS, S)$. Let ltp' be a local transaction program with fixed-structure such that execution of ltp' always results in a transaction with structure $struct(lt)$. ltp' may not be a correct transaction program. Due to the construction of ltp' and the gtp''s, and since $struct(gt^{GD}) = struct(gt')$, there exists a schedule S' which is the result of the execution of transaction programs in $\tau p'_G \cup \{ltp'\}$ from DS such that $struct(S'^{GD}) = struct((S^{\tau_G \cup lt})^{GD})$. Further, since S' contains only a subset of the transactions that S contains, S' is a G2LPL schedule.

As each site follows some serialization protocol, S^{D_i} is serializable. Let $\tau_1 = \tau_G \cup lt$ and $\tau_2 = \tau'_G \cup lt'$. Note that $\tau_w(GD_i, S) \subseteq \tau_1$ (since only global transactions write global data items) and $\tau_w(GD_i, S') \subseteq \tau_2$. Let $gt_1, \ldots, gt_r, lt, gt_{r+1}, \ldots, gt_n$ be a serialization order of global transactions in $(S^{\tau_1})^{GD_i}$ such that $lt \overset{*(S^{\tau_1})^{GD_i}}{\rightsquigarrow} gt_j$, for all j, $j = r+1, \ldots, n$, and the order $gt_1, gt_2, \ldots, gt_r$ is consistent with the serialization order of transactions in S^{τ_G}. In S', too, since $struct(S'^{GD}) = struct((S^{\tau_G \cup lt})^{GD})$, $gt'_1, \ldots, gt'_r, lt', gt'_{r+1}, \ldots, gt'_n$, is a serialization order of transactions in $(S'^{\tau_2})^{GD_i}$, where gt_i, gt'_i are global transactions resulting from the execution of gtp_i and gtp'_i respectively (note that $gt'_1, \ldots, gt'_r, lt', gt'_{r+1}, \ldots, gt'_n$, is a serialization order of transactions in S'^{GD_i}, since only lt accesses data items in $D_i - GD_i$). Let $p' \in lt'$ such that p and p' are corresponding operations. Since $struct(S'^{GD}) = struct((S^{\tau_G \cup lt})^{GD})$, $WS(after(gt_j, p, S)) = WS(after(gt'_j, p', S'))$, for all j, $j = 1, 2, \ldots, r$. In addition, since both

data items in d_i and $DS_1^{d_i}$ is consistent, $DS_2^{d_i}$ is consistent. By Lemma 3, $DS_2^{D-d_i}$ must be inconsistent. Let DS_3 be a database state such that $DS_3^{D-d_i} = DS_1^{D-d_i}$ and $DS_3^{d_i} = DS^{d_i}$. Let $\{DS_3\}tp'\{DS_4\}$. Since tp' does not access data items in d_i, $DS_4^{D-d_i} = DS_2^{D-d_i}$. Since $DS_2^{D-d_i}$ is inconsistent, $DS_4^{D-d_i}$ must be inconsistent. Since $d_i \cap d_j = \emptyset$, $i \neq j$, $DS_3^{d_i}$ is consistent, $DS_3^{D-d_i}$ is consistent, by Lemma 3, DS_3 is consistent. If $\{DS_3\}tp\{DS_5\}$, due to definition of $\{tp\}^{ds}$, $DS_5^{D-d_i} = DS_4^{D-d_i}$. Since tp preserves database consistency, and DS_3 is consistent, DS_5 must be consistent. Thus $DS_4^{D-d_i}$ is consistent. Contradiction. Hence tp' preserves database consistency. □

Lemma 17 *Let S be a G2LPL schedule consisting of correct transactions in the $G_{rw}L_r$ model. Let lt be a local transaction executing at site s_i, for some i, $i = 1, 2, \ldots, m$. If there are no integrity constraints between local and global data items, global transaction programs have fixed-structure, and $read(before(gt^{LD}, p, S))$ is consistent, for all gt, $gt \in \tau_G$, then*

- *If s_i is a 2PL site and p is the operation where lt releases its first lock, $read(lt^{GD_i})$ is consistent.*
- *If s_i is a non-2PL site, then $read(before(lt^{GD_i}, p, S))$ is consistent, where p is any arbitrary operation in lt.*

Proof: We first transform S to a schedule containing only lt in addition to global transactions, and then use Lemma 15. In the $G_{rw}L_r$ model, only global transactions are permitted to write global data items, and thus lt's read operations on global data items depend on the writes by the global transactions. However, since global transactions are permitted to read local data items, the values written by global transactions may depend on the values written by local transactions. However, if the global transactions are modified by incorporating the values returned by global transaction reads of local data items in S, into the global transaction programs itself, lt's reads of global data items are not affected by the deletion of other local transactions from the schedule S. We now explain the transformation of S to a G2LPL schedule S' such that lt's reads of global data items remain the same in both schedules and S' contains a single local transaction.

Every global transaction program $gtp \in \tau p_G$ is modified as follows. Let $gt \in \tau_G$ be the global transaction that results from the execution of global transaction program gtp in S. It

cuting at a site, S^{τ_G} would be non-serializable. If s_i is a 2PL site, then the above is true by Lemma 7. Thus, since $struct(S') = struct(S^{\tau_1})$ and both S and S' execute from DS, $read(before(gt'_j, p', S')) = read(before(gt_j, p, S))$, for all j, $j = 1, 2, \ldots, r$.

S^{D_i} and S'^{D_i} are both serializable and a serialization order of transactions serialized before lt, lt' in S^{D_i} and S'^{D_i} is $gt_1, gt_2, \ldots, gt_r$ and $gt'_1, gt'_2, \ldots, gt'_r$ respectively, such that gt_i and gt'_i result from the execution of transaction program gtp_i. Further, since $struct(S') = struct(S^{\tau_1})$, $WS(after(gt_k, p, S)) = WS(after(gt'_k, p', S'))$, for all k, $k = 1, 2, \ldots, r$. Thus by Lemma 14, $state(lt, D_i - \bigcup_{k=1}^{r} WS(after(gt_k, p, S_1)), S, DS) = state(lt', D_i - \bigcup_{k=1}^{r} WS(after(gt'_k, p, S')), S', DS)$.

Since S' is serializable , $gt'_1, gt'_2, \ldots, gt'_r$ are correct transactions , and DS is consistent, $state(lt', D, S', DS)$ is consistent. Hence, $state(lt, D_i - \bigcup_{k=1}^{r} WS(after(gt_k, p, S)), S, DS)$ is consistent. □

The above lemma considered a G2LPL schedule containing only one local transaction. In lemma 17, we show that a schedule containing more than one local transaction can be transformed into a schedule containing only one local transaction and the state seen by the local transaction in both schedules is the same. We first introduce certain transformations that can be applied to transaction programs. Further, we show that under certain conditions, if the original transaction program preserves database consistency, then so does the transformed program. Let tp be a transaction program, and ds, a set of data-value pairs. $\{tp\}^{ds}$ denotes the transaction program that results from

- deleting statements from tp in which d occurs on the left hand side of an assignment, for all $(d, v) \in ds$ and
- substituting v for all other occurrences of d in the program, for all $(d, v) \in ds$.

Lemma 16 *Let $IC = C_1 \wedge C_2 \wedge \ldots \wedge C_n$, where C_i is defined over data items in d_i such that $d_i \cap d_j = \emptyset$, $i \neq j$. Let DS be a database state such that DS^{d_i} is consistent, for some i, $i = 1, 2, \ldots, n$. If tp preserves database consistency, then $tp' = \{tp\}^{ds}$ also preserves database consistency, where $ds = DS^{d_i}$.*

Proof: Suppose tp' does not preserve database consistency. Thus, for some consistent database state DS_1 such that $\{DS_1\}tp'\{DS_2\}$, DS_2 is inconsistent. Since tp' does not access

Lemma 15 *Let S be a G2LPL schedule in an HDBMS consisting of global transactions and a local transaction lt executing at site s_i, for some i, $i = 1, 2, \ldots, m$. Let DS be a consistent database state such that $legal(DS, S)$. Let $gt_1, gt_2, \ldots, gt_r, lt, gt_{r+1}, \ldots, gt_n$ be a serialization order of transactions in S^{D_i} such that $lt \overset{*S^{D_i}}{\rightsquigarrow} gt_j$, for all j, $j = r+1, \ldots, n$, and the order $gt_1, gt_2, \ldots, gt_r$ is consistent with the serialization order of transactions in S^{τ_G} (gt_i cannot be serialized before gt_j in the local schedule if gt_j is serialized before gt_i in S^{τ_G}, $1 \leq i < j \leq r$, since this would violate the assumption that S^{τ_G} is serializable). If global transaction programs have fixed-structure, then $state(lt, D_i - \bigcup_{k=1}^{r} WS(after(gt_k, p, S)), S, DS)$ is consistent, where p is*

- *any arbitrary operation belonging to lt, if s_i is a non-2PL site.*
- *the operation where lt releases its first lock, if s_i is a 2PL site.*

Proof: We construct a new schedule S' which is serializable and in which lt sees the same state as the one it sees in S. We denote by gtp_i, the transaction program which on execution results in transaction gt_i. Let $\tau_1 = \tau - \{gt_{r+1}, \ldots, gt_n\}$ and ltp' be a local transaction program with fixed-structure such that execution of ltp' always results in a transaction with structure $struct(lt)$ (note that ltp' may be an incorrect transaction program). Let S' be a schedule generated by the execution of transaction programs in $\tau p_G \cup ltp' - \{gtp_{r+1}, \ldots, gtp_n\}$ from database state DS such that $struct(S') = struct(S^{\tau_1})$, where τp_G is the set of global transaction programs. (Note that such an S' exists since global transaction programs and ltp' have fixed-structure. S' can be generated by executing the operations belonging to transactions in the same order as in S^{τ_1}). Since S^{τ_G} is serializable, and the order $gt_1, gt_2, \ldots, gt_r$ is consistent with the serialization order of global transactions in S^{τ_G}, S' is serializable with serialization order $gt'_1, gt'_2, \ldots, gt'_r, lt'$, where $gt'_1, gt'_2, \ldots, gt'_r, lt'$ are transactions that result from the execution of transaction programs $gtp_1, gtp_2, \ldots, gtp_r, ltp'$ (reads and writes of transactions resulting from re-execution of the global transaction programs may change, thus generating different transactions).

Let $p' \in lt'$ be the operation corresponding to $p \in lt$ (lt and lt' have the same structure). Note that before p, gt_j, for all j, $j = 1, 2, \ldots, r$, could not have read a value written by any of $gt_{r+1}, \ldots, gt_n$. If s_i is a non-2PL site, then the above is trivially true, since if the above were not true, then due to forced conflicts between global subtransactions exe-

b: By Lemma 8, $RS(before(t_i^d, p, S)) \subseteq d - \bigcup_{j=1}^{i-1} WS(after(t_j, p, S))$. By the proof of (a), $state(t_i, d - \bigcup_{j=1}^{i-1} WS(after(t_j^d, p, S), S, DS)$ is consistent. Thus, $read(before(t_i^d, p, S))$ is consistent. Hence proved. □

Lemma 14 *Let S_1, S_2 be schedules, and $d \subseteq D$. Let $legal(DS_1, S_1)$ and $legal(DS_2, S_2)$ such that $DS_1^d = DS_2^d$. Let t, t' be transactions in S_1, S_2 and p, p' be operations belonging to S_1, S_2 respectively. Let $(S_1^{\tau_1})^d$ and $(S_2^{\tau_2})^d$ be serializable, where $\tau_w(d, S_1) \subseteq \tau_1$ and $\tau_w(d, S_2) \subseteq \tau_2$. Let a serialization order of transactions serialized before t, t' in $(S_1^{\tau_1})^d$ and $(S_2^{\tau_2})^d$ be $t_1, \ldots, t_r$ and $t'_1, \ldots, t'_r$ respectively, where both t_i and t'_i are transactions that result from the execution of transaction program tp_i. If $read(before(t_j, p, S_1)) = read(before(t'_j, p', S_2))$, and $WS(after(t_j, p, S_1)) = WS(after(t'_j, p', S_2))$, for all j, $j = 1, \ldots, r$, then $state(t, d - \bigcup_{j=1}^{r} WS(after(t_j, p, S_1)), S_1, DS_1) = state(t', d - \bigcup_{j=1}^{r} WS(after(t'_j, p', S_2)), S_2, DS_2)$.*

Proof: The proof is by induction on i, $i = 1, 2, \ldots, r$.

Basis ($i = 1$): $state(t_1, d, S_1, DS_1) = state(t'_1, d, S_2, DS_2) = DS_1^d = DS_2^d$.

Induction: Assume for $i = k$, $state(t_k, d - \bigcup_{j=1}^{k-1} WS(after(t_j, p, S_1)), S_1, DS_1) = state(t'_k, d - \bigcup_{j=1}^{k-1} WS(after(t'_j, p', S_2)), S_2, DS_2)$. We need to show that the above is true for $i = k + 1$.

$read(before(t_k, p, S_1)) = read(before(t'_k, p', S_2))$ and $WS(after(t_k, p, S_1)) = WS(after(t'_k, p', S_2))$.

By IH,

$state(t_k, d - \bigcup_{j=1}^{k-1} WS(after(t_j, p, S_1)), S_1, DS_1) = state(t'_k, d - \bigcup_{j=1}^{k-1} WS(after(t'_j, p', S_2)), S_2, DS_2)$.

Since t_k and t'_k result from the execution of transaction program tp_k, by Lemma 11,

$state(t_{k+1}, d - \bigcup_{j=1}^{k} WS(after(t_j, p, S_1)), S_1, DS_1) = state(t'_{k+1}, d - \bigcup_{j=1}^{k} WS(after(t'_j, p', S_2)), S_2, DS$

Thus, the lemma is true for $i = r$. Using an argument similar to the above argument, it can be shown that

$state(t, d - \bigcup_{j=1}^{r} WS(after(t_j, p, S_1)), S_1, DS_1) = state(t', d - \bigcup_{j=1}^{r} WS(after(t'_j, p', S_2)), S_2, DS_2)$.

□

As was illustrated in Example 7, G2LPL schedules may not be serializable if they contain local transactions. In the following lemma, we prove a property of G2LPL schedules containing a single local transaction (in addition to global transactions).

Let $d = \{a, b\}$, and $DS_1 = \{(a, -1), (b, -1), (c, -1)\}$. Execution of tp_1 from initial state DS_1 results in the following transaction:

$$t_1 : \quad w_1(a, 1) \quad r_1(c, -1)$$

Let $p = w_1(a, 1)$. The state resulting from the execution of t_1 from DS_1 is $DS_2 = \{(a, 1), (b, -1)\}$. Thus, though $DS_1^d \cup read(before(t_1, p, S))$ is consistent, $DS_2^{d-WS(after(t_1,p,S))} = \{(a, 1), (b, -1)\}$ is inconsistent. This is due to the fact that tp_1 is does not have fixed-structure. □

We now use Lemma 12 to develop assertions about the database state during the execution of schedules generated by the execution of fixed-structure transaction programs.

Lemma 13 *Let $IC = C_1 \wedge C_2 \wedge \cdots C_l$, where IC, C_i are defined over data items in D, d_i respectively such that $d_i \cap d_j = \emptyset$, $i \neq j$. Let p be an operation in a schedule S resulting from the execution of fixed-structure transaction programs. Let DS be a database state such that DS^d is consistent and legal(DS, S). Let $d = d_k$ for some $k = 1, 2, \ldots, l$, $\tau_w(d, S) \subseteq \tau'$, and $(S^{\tau'})^d$ be serializable with serialization order $t_1, t_2, \ldots, t_n$.*

a: *If $read(before(t_j, p, S))$, $j = 1, 2, \ldots, i-1$, is consistent, then $state(t_i, d - \bigcup_{j=1}^{i-1} WS(after(t_j^d, p, S))$, is consistent, for all i, $i = 1, 2, \ldots, n$*

b: *If $read(before(t_j, p, S))$, $j = 1, 2, \ldots, i-1$, is consistent, then $read(before(t_i^d, p, S))$ is consistent, for all i, $i = 1, 2, \ldots, n$*

Proof:

a: The proof is by induction on i.

Basis ($i = 1$): Trivial, as $state(t_1, d, S, DS) = DS^d$, which is consistent.

Induction: Assume true for $i = k$, that is, if $read(before(t_j, p, S))$, $j = 1, 2, \ldots, k-1$, is consistent, then for $d' = d - \bigcup_{j=1}^{k-1} WS(after(t_j^d, p, S))$, $state(t_k, d', S, DS)$ is consistent. We need to show the above to be true for $i = k + 1$.

By IH, we know that $state(t_k, d', S, DS)$ is consistent. By Lemma 8, $RS(before(t_k^d, p, S)) \subseteq d'$. Since $d_i \cap d_j = \emptyset$ and $read(before(t_k, p, S))$ is given to be consistent, $state(t_k, d', S, DS) \cup read(before(t_k, p, S))$ is consistent. Since transaction programs have fixed-structure, by Lemma 12, $state(t_{k+1}, d' - WS(after(t_k^d, p, S), S, DS)$ is consistent. Thus, $state(t_{k+1}, d - \bigcup_{j=1}^{k} WS(after(t_j^d, p, S))$, is consistent. Hence proved.

Lemma 11 *Let t and t' be transactions resulting from the execution of a transaction program tp (let S, S' denote the schedules containing t, t' respectively). Let p, p' be corresponding operations belonging to t and t' respectively. Let DS_1, DS_2 be database states and $d \subseteq D$ such that $legal(DS_1, t)$, $legal(DS_2, t')$ and $DS_1^d = DS_2^d$. Let $\{DS_1\}t\{DS_3\}$ and $\{DS_2\}t'\{DS_4\}$. If $read(before(t,p,S)) = read(before(t',p',S'))$, then $DS_3^{d-WS(after(t,p,S))} = DS_4^{d-WS(after(t',p',S'))}$.*

Proof: Since write operations are a function of read operations that precede them and t, t' result from the execution of the same program, if all the read operations of t, t' preceding p, p' are the same , the write operations preceding p, p' stay the same. Also, since $WS(after(t,p,S)) = WS(after(t',p',S'))$ and $DS_1^d = DS_2^d$, $DS_3^{d-WS(after(t,p,S))} = DS_4^{d-WS(after(t',p',S'))}$. □

Lemma 12 *Let t be a transaction in a schedule S resulting from the execution of a* fixed-structure *transaction program tp ($S = t$). Let DS_1 be a database state such that $\{DS_1\}t\{DS_2\}$, DS_1^d is consistent and $legal(DS_1, t)$. For any operation $p \in S$, if $DS_1^d \cup read(before(t,p,S))$ is consistent, then $DS_2^{d-WS(after(t,p,S))}$ is consistent.*

Proof: Let DS_3 be a consistent database state such that $DS_3^{d \cup RS(before(t,p,S))} = DS_1^d \cup read(before(t,p,S))$. Let $\{DS_3\}tp\{DS_4\}$. Let t' be the transaction and S' be the schedule resulting from the execution of tp from DS_3 (note that $S' = t'$). Since tp has fixed-structure, $struct(t') = struct(t)$. Thus, there exists an operation p' in S' such that p and p' are corresponding operations. Since $DS_3^{RS(before(t,p,S))} = read(before(t,p,S))$ and $struct(t') = struct(t)$, $read(before(t,p,S)) = read(before(t',p',S'))$. Thus, from Lemma 11, $DS_4^{d-WS(after(t',p',S'))} = DS_2^{d-WS(after(t,p,S))}$. Since tp preserves database consistency, DS_4 is consistent. Hence proved. □

In Lemma 12, if transaction programs do not have fixed-structure, then $struct(t')$ may not be equal to $struct(t)$. As a result, $WS(after(t',p',S'))$ may not be equal to $WS(after(t,p,S))$ and thus, $DS_2^{d-WS(after(t,p,S))}$ may not be consistent. This is illustrated by the following example.

Example 13 Consider a database containing data items $D = \{a, b, c\}$. Let $IC = (a > 0 \rightarrow b > 0) \wedge c > 0$. Consider the following transaction program, tp_1.

$$\begin{array}{lll} tp_1: & a := 1; & \\ & \textbf{if}(c > 0) \quad \textbf{then} & b := 1 \end{array}$$

Hence proved. □

Corollary 5 *Let S be a schedule such that S^d is a 2PL schedule, where $d \subseteq D$. Let transaction $t \in S^d$, operation $p \in S$ and $t_1, t_2, \ldots, t_r$ be a serialization order of transactions serialized before t in S^d. Then,*

a: $RS(before(t^d, p, S)) \subseteq d - \bigcup_{k=1}^{r} WS(after(t_k, p, S)) - \bigcup_{t' \in C(t,d,p,S)} WS(t')$, *and*

b: *If* $brelease(t^d, p, S)$, *then* $RS(t^d) \subseteq d - \bigcup_{k=1}^{r} WS(after(t_k, p, S))$.

Proof:

Trivial as $RS(before(t^d, p, S)) \subseteq LRS(before(t^d, p, S))$ and $RS(t^d) \subseteq LRS(t^d)$. □

In Appendix A, we developed conditions under which executions of transactions and schedules preserve consistency of a set of data items. One would wish to make similar assertions about the consistency of database states during the execution of transactions and schedules. For an arbitrary transaction making any assertion is difficult since all we know about a transaction is that as an atomic unit it is correct and nothing else. However, if we restrict transactions to those resulting from fixed-structured transaction programs, we can then make assertions about the states which exist during its execution. We first define the notion of *corresponding operations.*

Definition 9 *Let t, t' be two transactions resulting from the execution of a transaction program tp (the schedules containing t and t' are S and S' respectively). $p \in t$ and $p' \in t'$ are said to be corresponding operations if $RS(before(t, p, S)) = RS(before(t', p', S'))$ and $WS(after(t, p, S)) = WS(after(t', p', S'))$.* □

Note that corresponding operations may not exist for a pair of transactions resulting from the execution of an arbitrary transaction program *tp* since the structure of the transaction may change depending on the database state it executed from. However, for transactions resulting from the execution of a fixed-structure transaction program, corresponding operations always exist. G2LPL preserves strong correctness of schedules only if global transaction programs have fixed-structure. In the proof we shall require certain properties of transaction programs with fixed-structure which are developed in the following lemmas.

Lemma 9 *Let S be a schedule and $d \subseteq D$ such that S^d is a 2PL schedule. Let $t_1, t_2, \ldots, t_n$ be a serialization order of transactions in S^d and p be an operation in S.*
$LRS(before(t_i^d, p, S)) \subseteq d - \bigcup_{j=1}^{i-1} WS(after(t_j^d, p, S))$, for all i, $i = 1, 2, \ldots, n$.

Proof: Suppose $LRS(before(t_i^d, p, S)) \not\subseteq d - \bigcup_{j=1}^{i-1} WS(after(t_j^d, p, S))$, for some i, $i = 1, 2, \ldots, n$. Thus, t_i obtains a read lock on a data item in d before t_j writes the data item, for some j, $j = 1, 2, \ldots, i-1$. Further, since S^d is a 2PL schedule, t_i must read the data item before t_j writes the data item and thus, must be serialized before t_j in S^d. Contradiction. Hence proved. □

Lemma 10 *Let S be a schedule such that S^d is a 2PL schedule, where $d \subseteq D$. Let transaction $t \in S^d$, operation $p \in S$ and $t_1, t_2, \ldots, t_r$ be a serialization order of transactions serialized before t in S^d. Then,*

a: *$LRS(before(t^d, p, S)) \subseteq d - \bigcup_{k=1}^{r} WS(after(t_k, p, S)) - \bigcup_{t' \in C(t,d,p,S)} WS(t')$, and*

b: *If $brelease(t^d, p, S)$, then $LRS(t^d) \subseteq d - \bigcup_{k=1}^{r} WS(after(t_k, p, S))$.*

Proof:

a: From Lemma 9, it follows that $LRS(before(t^d, p, S)) \subseteq d - \bigcup_{k=1}^{r} WS(after(t_k, p, S))$. Further, we need to show that, before p, t^d cannot obtain a lock for data items written by transactions in $C(t, d, p, S)$. Consider a transaction $t' \in C(t, d, p, S)$, and a data item $d' \in WS(t') \cap d$. If t were to lock d' for reading before t' obtained the write lock on it, t' would be serialized after t. If t obtained a read lock after the write lock was obtained on d' by t', then since S^d is a 2PL schedule and t' releases its first lock after p, t would have obtained the lock after p. Thus, $LRS(before(t^d, p, S)) \subseteq d - \bigcup_{k=1}^{r} WS(after(t_k, p, S)) - \bigcup_{t' \in C(r)} WS(t')$.

b: Since S^d is a 2PL schedule, S^d is serializable. By Lemma 9, $LRS(before(t^d, p, S)) \subseteq d - \bigcup_{k=1}^{r} WS(after(t_k, p, S))$. Let $d' \in (\bigcup_{k=1}^{r} WS(after(t_k, p, S_1))) \cap d$. After p, t cannot lock d' before any of $t_1, \ldots, t_r$ write on d' (since t is serialized after $t_1, \ldots, t_r$). Also, after p, t cannot read d' after any of $t_1, \ldots, t_r$ write on d' since t releases its first lock before p and since S^d is a 2PL schedule, t cannot obtain any more locks. Thus, $LRS(t^d) \subseteq d - \bigcup_{k=1}^{r} WS(after(t_k, p, S))$.

Lemmas 12 through 14 prove properties of serializable schedules consisting of transactions with fixed-structure. In order to show a schedule is strongly correct, we need to show that both local and global transactions read consistent data items. Local transactions are shown to read consistent data in Lemma 17 using lemmas 15 and 16. In Lemma 18 and Lemma 19, global transactions are shown to read consistent data items. Finally, Lemma 20 combines results from Lemma 17 and Lemma 19, and is used in the proof of the theorem.

Lemma 7 *Let S be a G2LPL schedule and $t \in S^{D_i}$, where s_i is a 2PL-site. Let $t_1, t_2, \ldots, t_r, t, t_{r+1}, \ldots, t_n$ be a serialization order of transactions in S^{D_i} such that $t \overset{*S^{D_i}}{\rightsquigarrow} t_j$, for all j, $j = r+1, \ldots, n$. Let p be the operation in S^{D_i} where t releases its first lock and $o \in t_k$, for some k, $k = r+1, \ldots, n$. If $o \overset{*S^{\tau_G}}{\rightsquigarrow} o'$, where $o' \in t'$, then $o' \notin before(t', p, S)$.*

Proof: Since $o \overset{*S^{\tau_G}}{\rightsquigarrow} o'$ and S is a G2LPL schedule, o' cannot be scheduled for execution unless t_k releases a global lock. Also, t_k does not release a global lock unless it has obtained all its global locks, and local locks at the 2PL sites. Since $t \overset{*S^{D_i}}{\rightsquigarrow} t_k$, and s_i is a 2PL site, t_k gains all its local locks at site s_i only after t releases its first lock at site s_i. Thus, o' cannot execute before p. Thus, $o' \notin before(t', p, S)$. □

The following lemma states a simple property of serializable schedules. Before an operation p during the execution of a serializable schedule if a transaction t reads (or writes) a data item, that data item cannot be written at a later time by a transaction that is serialized before t.

Lemma 8 *Let S be a schedule and $d \subseteq D$ such that S^d is serializable. Let $t_1, t_2, \ldots, t_n$ be a serialization order of transactions in S^d and p be an operation in S.*
$RS(before(t_i^d, p, S)) \subseteq d - \bigcup_{j=1}^{i-1} WS(after(t_j^d, p, S))$, for all i, $i = 1, 2, \ldots, n$.

Proof: Suppose $RS(before(t_i^d, p, S)) \not\subseteq d - \bigcup_{j=1}^{i-1} WS(after(t_j^d, p, S))$, for some i, $i = 1, 2, \ldots, n$. Thus, $before(t_i^d, p, S)$ reads a data item in d written by $after(t_j, p, S)$, for some j, $j = 1, 2, \ldots, i-1$. In that case t_i must be serialized before t_j in S^d since $before(t_i, p, S)$ precedes $after(t_j, p, S)$ in S^d. Contradiction. Hence proved. □

A similar result for 2PL schedules is stated in the following lemma.

Consider a schedule S and $d \subseteq D$ such that S^d is a 2PL schedule. Let t be a transaction in S^d, p be an operation in S, and $t_1, t_2, \ldots, t_n$ be a serialization order of transactions serialized before t in S^d. $C(t,d,p,S)$ is defined as follows.

$$C(t,d,p,S) = \{t' : t' \in \{t_1, t_2, \ldots, t_n\} \wedge t'^d \text{ releases its first lock in } S \text{ after } p\}$$

We define the following for seq, which is a subsequence of S. $brelease(seq,p,S)$ is true iff the first unlock operation in seq precedes p, where p is an operation in S. $LRS(seq)$ denotes the set of data items for which read locks have been obtained by seq.

$$LRS(seq) = \{y : o \in seq \wedge y = entity(o) \wedge action(o) = lr\}$$

Thus, $LRS(before(seq,p,S))$ denotes the set of data items for which locks have been obtained by seq before operation p in the schedule S. $Lread(seq)$ denotes the database state seen by seq as a result of reading data items for which it has obtained read locks.

$$Lread(seq) = \{(y,z) : o \in seq \wedge y = entity(o) \wedge z = value(o) \wedge y \in LRS(seq) \wedge action(o) = r\}$$

Note that LRS and $Lread$ are only defined for sequences of operations that access data items at a 2PL site. Thus, for a transaction t, $LRS(t) \neq RS(t)$, since no locks need to be obtained by transactions accessing data items at non-2PL sites. However, if for $d \subseteq D$, S^d is a 2PL schedule, $RS(t^d) = LRS(t^d)$ and, $read(t^d) = Lread(t^d)$, since transaction t^d must obtain read lock on a data item before reading the data item. Let DS is a database state such that $legal(DS,S)$. Since $read(t^d) \subseteq state(t,d,S,DS)$, $Lread(t^d) \subseteq state(t,d,S,DS)$. Also, $RS(before(t,p,S)) \subseteq LRS(before(t,p,S))$, where p is an operation belonging to schedule S. In general, for a sequence of operations, seq, if $RS(seq) = LRS(seq)$, then $RS(seq^d) = LRS(seq^d)$, and if $RS(seq) \subseteq LRS(seq)$, then $RS(before(seq,p,S)) \subseteq LRS(before(seq,p,S))$. However, it must be noted that $RS(after(t,p,S)) \not\subseteq LRS(after(t,p,S))$.

We need to differentiate between 2PL sites and non-2PL sites in the proof. We assume, that the HDBMS consists of sites $s_1, s_2, \ldots, s_m, s_{(m+1)}, s_{(m+2)}, \ldots, s_{m+q}$, where $s_1, s_2, \ldots, s_m$ are 2PL-sites and $s_{(m+1)}, s_{(m+2)}, \ldots, s_{m+q}$ are non-2PL sites. The set of data items at the 2PL sites, $D1 = \bigcup_{i=1}^{m} D_i$, and the set of data items at the non-2PL sites, $D2 = \bigcup_{i=1}^{q} D_{m+i}$.

Outline of Proof We give a brief outline of the structure of the proof and the various lemmas used in the proof. In lemmas 7 through 10, we state important properties of G2LPL schedules.

-Appendix B-

We prove Theorem 8 in this appendix. For the proof, we shall need to enhance our notation substantially. Since we now deal with schedules that are produced (at least in part) by LTMs which follow a locking protocol, we generalize our definition of transactions and schedules. Transactions and schedules, besides containing read and write operations, also contain lock and unlock operations. The lock and unlock operations correspond to the local lock and unlock operations at the 2PL sites and not the global lock and unlock operations. A lock operation is a two-tuple $(action(o_i), entity(o_i))$, where $action(o_i)$ is either a read lock (lr), or a write lock (lw), and $entity(o_i)$ is the data item being locked. The unlock operation is similarly defined. We assume that transactions access every data item for which they obtain locks. We now develop the notation needed to prove Theorem 8.

Let seq be a subsequence of schedule S and p be an operation in S. We denote by $before(seq, p, S)$, the subsequence of seq consisting of all the operations that precede p in S. If p belongs to seq, then $before(seq, p, S)$ includes p. The subsequence of seq consisting of all the operations not in $before(seq, p, S)$ is denoted by $after(seq, p, S)$. The number of operations preceding operation p in schedule S is denoted by $depth(p, S)$. In Example 10, if $p = w_2(a, 10)$, then

$$before(t_2, p, S) = \quad r_2(d, 10) \quad w_2(a, 10)$$
$$after(t_1, p, S) = \quad r_1(c, 5) \quad w_1(b, 5)$$
$$depth(p, S) = 2.$$

We now define the notion of conflict between operations and transactions.

Definition 8 *Let o_1 and o_2 be operations in schedule S. $o_1 \overset{S}{\rightsquigarrow} o_2$ if*

- *for some transaction $t \in S$, $o_1, o_2 \in t$ and $o_1 \prec_t o_2$, or*
- *for some transactions $t_1, t_2 \in S$, such that $t_1 \neq t_2$, $o_1 \in t_1$ and $o_2 \in t_2$, $entity(o_1) = entity(o_2)$, $(action(o_1) = W$ or $action(o_2) = W)$ and $o_1 \prec_S o_2$.*

The notion of conflict can be extended to transactions as follows. If $o_1 \in t_1$, $o_2 \in t_2$, where t_1 and t_2 are transactions in schedule S, and $o_1 \overset{S}{\rightsquigarrow} o_2$, then $t_1 \overset{S}{\rightsquigarrow} t_2$. $\overset{*S}{\rightsquigarrow}$ is the transitive closure of $\overset{S}{\rightsquigarrow}$. □

items, the integrity constraints can be viewed as $IC = L_1 \wedge \ldots \wedge L_m \wedge G$, where L_i is a conjunct defined over data items in LD_i and G is a conjunct defined over data items in GD.

We use the fact that transaction programs are LDP to prove that $L_1, L_2, \ldots, L_n$ are preserved by S and global transactions read consistent local data items. Since DS_1 is consistent, by Lemma 3, $DS_1^{D_i}$ is consistent. Since S^{D_i} is serializable (let $t_1, t_2, \ldots, t_n$ be a serialization order of transactions in S^{D_i}) and transaction programs are LDP, by Lemma 6, $DS_2^{D_i}$ is consistent and $state(t_j, D_i, S, DS_1)$, for all j, $j = 1, 2, \ldots, n$, is consistent. Since $LD_i \subseteq D_i$ and $DS_2^{D_i}$ is consistent, $DS_2^{LD_i}$ is consistent. Since $read(t^{D_i}) \subseteq state(t, D_i, S, DS_1)$, $read(t^{D_i})$ is consistent, for all t, $t \in \tau$. Since $LD_i \subseteq D_i$, $read(t^{LD_i})$, for all t, $t \in \tau_G$, is consistent. Also, since local transactions access data items at a single site, $read(t)$, for all t, $t \in \tau_L$, is consistent.

We now show that G is preserved by S. Since only global transactions write on global data items $\tau_w(GD, S) \subseteq \tau_G$. Since S^{τ_G} is serializable $(S^{\tau_G})^{GD}$ is serializable. As shown above, $read(t^{LD_i})$ is consistent, $t \in \tau_G$. Since, $LD_i \cap LD_j = \emptyset$, $i \neq j$, by Lemma 2, $\bigcup_{i=1}^{m} read(t^{LD_i}) = read(t^{\cup_{i=1}^{m} LD_i})$ is consistent. Since DS_1 is consistent, by Lemma 3, DS_1^{GD} is consistent. Thus, by Corollary 4, DS_2^{GD} is consistent and $state(t, GD, S, DS_1)$, for all t, $t \in \tau_G$ is consistent. Since $read(t^{GD}) \subseteq state(t, GD, S, DS_1)$ and $read(t^{\cup_{i=1}^{m} LD_i})$, $t \in \tau_G$, is consistent, by Lemma 2, $read(t)$, for all t, $t \in \tau_G$, is consistent.

Thus, DS_2^{GD} and $DS_2^{LD_i}$, for all i, $i = 1, 2, \ldots, m$ is consistent. Hence, by Lemma 3, DS_2 is consistent. Thus, S is strongly correct. $\square$

Thus, DS_2^{GD} and $DS_2^{D_i}$ for all i, $i = 1, 2, \ldots, m$ is consistent. Hence, by Lemma 3, DS_2 is consistent. Thus, S is strongly correct. □

In order to prove Theorem 4, we use a property of *predicate wise serializable (PWSR)* schedules developed in [10]. Before proceeding, we first define PWSR, and then state the property of PWSR schedules.

Definition 7 *Let $IC = C_1 \wedge C_2 \wedge \cdots \wedge C_l$, where IC, C_i are defined over data items in D, d_i respectively. A schedule S is PWSR if S^{d_i} is serializable, for all i, $i = 1, 2, \ldots, l$ [8].* □

Any serializable schedule is PWSR, but not vice-versa.

Theorem 9 *Let $IC = C_1 \wedge C_2 \wedge \ldots \wedge C_l$, where IC, C_i are defined over data items in D, d_i respectively such that $d_i \cap d_j = \emptyset$, $i \neq j$. Let S be a schedule consisting of transactions resulting from the execution of fixed-structure transaction programs. If S is a PWSR schedule, then it is strongly correct [10].* □

We now prove Theorem 4.

Proof of Theorem 4 (G_{rw} Model): As no integrity constraints are present between local and global data items, the integrity constraints can be viewed as $IC = L_1 \wedge \ldots \wedge L_m \wedge G$, where L_i is a conjunct defined over data items in LD_i and G is a conjunct defined over data items in GD. As a result of the model, $LD_i \cap LD_j = \emptyset$ and $LD_i \cap GD = \emptyset$.

Since only global transactions access global data items and S^{τ_G} is serializable, S^{GD} is serializable. Since S^{D_i} is serializable , S^{LD_i} is serializable for all i, $i = 1, 2, \ldots, m$. Thus S is a PWSR schedule. By Theorem 9, S is strongly correct. □

Proof of Theorem 5 ($G_{rw}L_r$ Model): Let DS_1 be a consistent database state such that $legal(DS_1, S)$. Let $\{DS_1\}S\{DS_2\}$. In order to prove that S is strongly correct, we need to show that DS_2 is consistent. Since no integrity constraints are present between local and global data

Basis ($j = 1$): $state(t_1, D_i, S, DS_1) = DS_1^{D_i}$, which is given to be consistent.
Induction: Assume $state(t_k, D_i, S, DS_1)$ is consistent. Let DS_3 be a database state such that $DS_3^{D_i} = state(t_k, D_i, S, DS_1)$ and $legal(DS_3, t_k)$. By IH, $DS_3^{D_i}$ is consistent. Let $\{DS_3\}t_k\{DS_4\}$. Since, transaction programs are LDP, $DS_4^{D_i}$ is consistent. As $DS_4^{D_i} = state(t_{k+1}, D_i, S, DS_1)$, $state(t_{k+1}, D_i, S, DS_1)$ is consistent.

As shown above, $state(t_n, D_i, S, DS_1)$ is consistent. Thus, (using a similar argument as above) $DS_2^{D_i}$ is consistent. □

It must be noted that consistency of all the local database states does not imply the consistency of the global database state. Thus, a schedule S resulting from the execution of LDP programs such that S^{D_i} is serializable, $i = 1, 2, \ldots, m$, would leave all the local databases in a consistent state if they were initially consistent. However, S may not preserve global database consistency.

Proof of Theorem 3 (L_r Model): Let DS_1 be a consistent database state such that $legal(DS_1, S)$. Let $\{DS_1\}S\{DS_2\}$. In order to show that S is strongly correct, we need to show that DS_2 is consistent. The integrity constraints can be viewed as $IC = C_1 \wedge \cdots \wedge C_m \wedge G$, where C_i is a conjunct defined over data items in D_i and G is a conjunct defined over data items in GD. We show that $C_1, C_2, \ldots, C_m$ are preserved by S if transaction programs are LDP. Since DS_1 is consistent, by Lemma 3, $DS_1^{D_i}$ is consistent. Since S^{D_i} is serializable and transaction programs are LDP, by Lemma 6, $DS_2^{D_i}$ is consistent and $state(t, D_i, S, DS_1)$, for all t, $t \in \tau$, is consistent. Furthermore, since local transactions access data items at a single site, $read(t) \subseteq state(t, D_i, S, DS_1)$, $t \in \tau_L$. Thus, $read(t)$, for all t, $t \in \tau_L$ is consistent.

G can now be shown to be preserved by S using Corollary 3 as follows. Since only global transactions write global data items, $\tau_w(GD, S) \subseteq \tau_G$. Since S^{τ_G} is serializable, $(S^{\tau_G})^{GD}$ is serializable. Since global transactions read only data items in GD, $RS(t) \subseteq GD$, where $t \in \tau_w(GD, S)$. By Lemma 3, since DS_1 is consistent, DS_1^{GD} is consistent. Hence, by Corollary 3, DS_2^{GD} is consistent and $state(t, GD, S, DS_1)$, for all t, $t \in \tau_G$, is consistent. Since global transactions only read data items in GD, $read(t) \subseteq state(t, GD, S, DS_1)$, $t \in \tau_G$. Thus, $read(t)$, for all t, $t \in \tau_G$, is consistent.

Lemma 3, since DS_1 is consistent, $DS_1^{LD_i}$ is consistent. Hence by Corollary 3, $DS_2^{LD_i}$ is consistent and $state(t_j, LD_i, S, DS_1)$, for all j, $j = 1, 2, \ldots, n$ is consistent. Since $read(t^{LD_i}) \subseteq state(t, LD_i, S, DS_1)$, $read(t^{LD_i})$ is consistent, for all t, $t \in \tau$. Since local transactions access data items at a single site $read(t)$, for all t, $t \in \tau_L$, is consistent.

G can now be shown to be preserved by S using Corollary 4 as follows. Since only global transactions write on global data items $\tau_w(GD, S) \subseteq \tau_G$. Since S^{τ_G} is serializable, $(S^{\tau_G})^{GD}$ is serializable. As shown above, $read(t^{LD_i})$ is consistent, $t \in \tau_G$. Since, $LD_i \cap LD_j = \emptyset$, $i \neq j$, by Lemma 2, $\bigcup_{i=1}^{m} read(t^{LD_i}) = read(t^{\cup_{i=1}^{m} LD_i})$, $t \in \tau_G$, is consistent. Since DS_1 is consistent, by Lemma 3, DS_1^{GD} is consistent. Also, $GD \cap LD_i = \emptyset$, for all i, $i = 1, 2, \ldots, m$. Thus, by Corollary 4, DS_2^{GD} is consistent and $state(t, GD, S, DS_1)$, for all t, $t \in \tau_G$ is consistent. Since $read(t^{GD}) \subseteq state(t, GD, S, DS_1)$ and $read(t^{\cup_{i=1}^{m} LD_i})$, $t \in \tau_G$, is consistent, by Lemma 2, $read(t)$, for all t, $t \in \tau_G$, is consistent.

Thus, DS_2^{GD} and $DS_2^{LD_i}$, for all i, $i = 1, 2, \ldots, m$ is consistent. Hence, by Lemma 3, DS_2 is consistent. Thus, S is strongly correct. $\Box$

In the L_r model transaction programs were restricted to be LDP. LDP programs, if executed from a consistent local database state leave the local database in a consistent state. Before proving Theorem 3 we develop Lemma 6, and show by a simple induction argument that a serializable local schedule resulting from the execution of LDP programs leaves the local database in a consistent state if the local database was initially consistent.

Lemma 6 *Let S be a schedule and DS_1 be a database state such that $legal(DS_1, S)$ and $\{DS_1\}S\{DS_2\}$. If, for some i, $i = 1, 2, \ldots, m$,*

- *S^{D_i} is serializable (let $t_1, \ldots, t_n$ be a serialization order of transactions in S^{D_i}),*
- *transactions programs are LDP, and*
- *$DS_1^{D_i}$ is consistent,*

then $DS_2^{D_i}$ is consistent and $state(t_j, D_i, S, DS_1)$ is consistent, for all j, $j = 1, 2, \ldots n$.

Proof: We begin by showing that $state(t_j, D_i, S, DS_1)$, for all j, $j = 1, 2, \ldots, n$ is consistent. The proof is by induction on j.

then DS_2^d *is consistent and* $state(t_i, d, S, DS_1)$ *is consistent for all* i, $i = 1, 2, \ldots, n$.

Proof: Follows directly from Lemma 5 as for every transaction $t \in \tau_w(d, S)$, since $RS(t) \subseteq d$, $read(t) \subseteq state(t, d, S, DS_1)$. Thus, whenever $state(t, d, S, DS_1)$ is consistent, $state(t, d, S, DS_1) \cup read(t)$ is consistent. □

Corollary 4 *Let* $IC = C_1 \wedge C_2 \wedge \cdots \wedge C_l \wedge C$, *where* $C_1, C_2, \ldots, C_l, C$ *are defined over data items in* $d_1, d_2, \ldots, d_l, d$ *such that* $d_i \cap d = \emptyset$, $i = 1, 2, \ldots, l$. *Let* S *be a schedule and* DS_1 *be a database state such that* $legal(DS_1, S)$ *and* $\{DS_1\}S\{DS_2\}$. *Let* τ' *be any set of transactions such that* $\tau_w(d, S) \subseteq \tau'$. *If*

- $(S^{\tau'})^d$ *is serializable (let* $t_1, t_2, \ldots, t_n$ *be a serialization order of transactions in* $(S^{\tau'})^d$*),*
- $read(t^{d'})$ *is consistent, for all* t, $t \in \tau_w(d, S)$ *and* $d' = \bigcup_{i=1}^{l} d_i$, *and*
- DS_1^d *is consistent,*

then DS_2^d *is consistent and* $state(t_i, d, S, DS_1)$ *is consistent for all* i, $i = 1, 2, \ldots, n$.

Proof: For every $t \in \tau_w(d, S)$, $read(t^{d'})$ is given to be consistent. Thus, whenever $state(t, d, S, DS_1)$ is consistent, since $d' \cap d = \emptyset$ and $read(t^d) \subseteq state(t, d, S, DS_1)$, $state(t, d, S, DS_1) \cup read(t)$ is consistent. Hence, the result trivially follows from Lemma 5. □

Proof of Theorem 2 (G_r Model): Let DS_1 be a consistent database state such that $legal(DS_1, S)$. Let $\{DS_1\}S\{DS_2\}$. In order to show that S is a strongly correct, we need to show that DS_2 is consistent. As no integrity constraints are present between local and global data items, the integrity constraints can be viewed as $IC = L_1 \wedge \cdots \wedge L_m \wedge G$, where L_i is a conjunct defined over data items in LD_i and G is a conjunct defined over data items in GD. As a result of the model, $LD_i \cap LD_j = \emptyset$, $i \neq j$, and $LD_i \cap GD = \emptyset$.

We now use Corollary 3 to show that $L_1, L_2, \ldots, L_m$ are preserved by S and global transactions read consistent local data items. Since only local transactions at site s_i write local data items at site s_i, $\tau_w(LD_i, S) \subseteq \tau_L \subseteq \tau$. Since $S^\tau = S$ and S^{D_i} is serializable, $(S^\tau)^{LD_i}$ is serializable. Let $t_1, t_2, \ldots, t_n$ be a serialization order of transactions on $(S^\tau)^{LD_i}$. Since local transactions at site i read only from LD_i, $RS(t) \subseteq LD_i$, where $t \in \tau_w(LD_i, S)$. By

Lemma 5 *Let S be a schedule DS_1 be a database state such that $legal(DS_1, S)$ and $\{DS_1\}S\{DS_2\}$. Let $d \subseteq D$ and τ' be any set of transactions such that $\tau_w(d, S) \subseteq \tau'$. If*

- *$(S^{\tau'})^d$ is serializable (let $t_1, t_2, \ldots, t_n$ be a serialization order of transactions in $(S^{\tau'})^d$),*
- *if $state(t, d, S, DS_1)$ is consistent, then $state(t, d, S, DS_1) \cup read(t)$ is consistent, for all t, $t \in \tau_w(d, S)$, and*
- *DS_1^d is consistent,*

then DS_2^d is consistent and $state(t_i, d, S, DS_1)$ is consistent for all i, $i = 1, 2, \ldots, n$.

Proof: The proof is by induction on i.

Basis ($i = 1$): $state(t_1, d, S, DS_1) = DS_1^d$, which is given to be consistent.

Induction: Assume $state(t_k, d, S, DS_1)$ is consistent. We need to show to show $state(t_{k+1}, d, S, DS_1)$ is consistent. Consider two cases:

Case 1 ($t_k \notin \tau_w(d, S)$): Since $t_k \notin \tau_w(d, S)$, it follows from the definition of τ_w that $WS(t_k) \cap d = \emptyset$. Thus, $state(t_{k+1}, d, S, DS_1) = state(t_k, d, S, DS_1)$. By IH, $state(t_k, d, S, DS_1)$ is consistent. Thus $state(t_{k+1}, d, S, DS_1)$ is consistent as well.

Case 2 ($t_k \in \tau_w(d, S)$): By IH, $state(t_k, d, S, DS_1)$ is consistent. Thus, $state(t_k, d, S, DS_1) \cup read(t_k)$ is consistent. Hence, there exists a consistent database state, DS_3, such that $DS_3^d = state(t_k, d, S, DS_1)$ and $DS_3^{RS(t_k)} = read(t_k)$. Thus, $legal(DS_3, t_k)$. Let $\{DS_3\}t_k\{DS_4\}$. As $DS_3^d \cup read(t_k) =$ is consistent, by Lemma 4, DS_4^d is consistent. Since $DS_4^d = state(t_{k+1}, d, S, DS_1)$, $state(t_{k+1}, d, S, DS_1)$ is consistent.

As shown above, $state(t_n, d, S, DS_1)$ is consistent. Thus, by Lemma 4, (using a similar argument as above) DS_2^d is consistent. □

For the proofs of our theorems we shall require the following corollaries which follow from Lemma 5.

Corollary 3 *Let S be a schedule, and DS_1 be a database state such that $legal(DS_1, S)$ and $\{DS_1\}S\{DS_2\}$. Let $d \subseteq D$ and τ' be any set of transactions such that $\tau_w(d, S) \subseteq \tau'$. If*

- *$(S^{\tau'})^d$ is serializable (let $t_1, t_2, \ldots, t_n$ be a serialization order of transactions in $(S^{\tau'})^d$),*
- *$RS(t) \subseteq d$, for all t, $t \in \tau_w(d, S)$, and*
- *DS_1^d is consistent,*

$i = 1, 2, \ldots, n$. Thus, $DS \models C_1 \wedge C_2 \wedge \cdots \wedge C_n$. Hence DS is consistent. □

The following lemma states the conditions under which a transaction, on execution, leaves the database in a consistent state.

Lemma 4 *Let t be a transaction and $d \subseteq D$. Let DS_1 be a database state such that $legal(DS_1, t)$ and $\{DS_1\}t\{DS_2\}$. If $DS_1^d \cup read(t)$ is consistent, then DS_2^d is consistent.*

Proof: By the definition of consistency, there exists a consistent state DS_3 such that $DS_3^{d \cup RS(t)} = DS_1^d \cup read(t)$. Thus, $legal(DS_3, t)$. Let $\{DS_3\}t\{DS_4\}$. Since t preserves database consistency, DS_4 is consistent. Also, since $DS_1^d = DS_3^d$, $DS_2^d = DS_4^d$. Thus, DS_2^d is consistent. □

Note that Lemma 4 requires $DS_1^d \cup read(t)$ to be consistent if DS_2^d is to be consistent. Consistency of DS_1^d and $read(t)$ does not ensure consistency of DS_2^d since $DS_1^d \cup read(t)$ may not be consistent. As a result, even if DS_1^d and $read(t)$ are consistent, a consistent state DS_3 such that $DS_3^{d \cup RS(t)} = DS_1^d \cup read(t)$ may not exist.

Example 12 Consider a database with data items $D = \{a, b, c\}$. Let $IC = (a = b \wedge b = c)$. Consider the following transaction program tp_1.

$$tp_1 : \quad a := c$$

Let $d = \{a, b\}$ and $DS_1 = \{(a, -1), (b, -1), (c, 1)\}$. Consider the execution of transaction program tp_1 from DS_1 that results in the following transaction:

$$t_1 : \quad r_1(c, 1) \quad w_1(a, 1)$$

The database state DS_2 resulting from the execution of tp_1 from DS_1 is $\{(a, 1), (b, -1), (c, 1)\}$. Thus, even though DS_1^d and $read(t_1)$ are consistent, their union $\{(a, -1), (b, -1), (c, 1)\}$ is inconsistent and as a result, $DS_2^d = \{(a, 1), (b, -1)\}$ is inconsistent. □

Earlier, in Lemma 4 we specified conditions required to ensure that execution of a transaction preserves consistency of a set of data items. We now use Lemma 4 to develop conditions under which schedules preserve consistency of a set of data items.

Proof: $\Leftarrow$:

If $\bigcup_{i=1}^{n} DS^{d_i'}$ is consistent, then $DS^{d_i'}$, $i = 1, 2, \ldots, n$ is consistent . This follows directly from the definition of database consistency.

$\Rightarrow$:

We now prove that if $DS^{d_i'}$, $i = 1, 2, \ldots, n$ is consistent, then $\bigcup_{i=1}^{n} DS^{d_i'}$ is consistent. Since $DS^{d_i'}$ are consistent, there exist consistent database states, DS_i such that $DS_i^{d_i'} = DS^{d_i'}$, $i = 1, 2, \ldots, n$. Let DS_0 be a database state such that $DS_0^{d_i} = DS_i^{d_i}$, for all i, $i = 1, 2, \ldots, n$ (such a DS_0 exists since $d_i \cap d_j = \emptyset, i \neq j$). $DS_0 \models C_i$, for all i, $i = 1, 2, \ldots, n$, since $DS_i \models C_i$ (DS_i is consistent), $DS_0^{d_i} = DS_i^{d_i}$ and C_i is defined only over data items in d_i. Thus, $DS_0 \models C_1 \wedge C_2 \wedge \cdots \wedge C_n$. Also, $\bigcup_{i=1}^{n} DS^{d_i'} \subseteq DS_0$. Thus, there exists a consistent database state DS_0 such that $\bigcup_{i=1}^{n} DS^{d_i'} \subseteq DS_0$. Hence, by definition of database consistency, $\bigcup_{i=1}^{n} DS^{d_i'}$ is consistent. $\square$

Note that it is essential for the data items, over which conjuncts are defined, to be disjoint if Lemma 2 is to hold. For example, let $IC = ((a = 5 \rightarrow b = 5) \wedge (c = 5 \rightarrow b = 6))$. Let $d_1' = \{a\}$, $d_2' = \{c\}$, $DS^{d_1'} = \{(a, 5)\}$ and $DS^{d_2'} = \{(c, 5)\}$. Thus, even though $DS^{d_1'}$ and $DS^{d_2'}$ are consistent, $DS^{d_1'} \cup DS^{d_2'}$ is inconsistent. Since $d_i \cap d_j \neq \emptyset$, database state DS_0 in the proof of Lemma 2 may not exist.

We now show that if a database state restricted to the data items in every conjunct is consistent, then the database state itself must be consistent. In contrast to Lemma 2, the sets of data items over which the C_i's are defined are not required to be disjoint.

Lemma 3 *Let $IC = C_1 \wedge C_2 \wedge \cdots \wedge C_n$, where $C_1, C_2, \ldots, C_n$ are defined over data items in $d_1, d_2, \ldots, d_n$. Let DS be a database state. DS^{d_i}, for all i, $i = 1, 2, \ldots, n$ is consistent iff DS is consistent.*

Proof:

$\Leftarrow$:

Follows directly from the definition of consistency.

$\Rightarrow$:

Since DS^{d_i} is consistent and C_i is defined only over data items in d_i, $DS \models C_i$, for all i,

from database state $DS_1 = \{(a,0),(b,10),(c,5),(d,10)\}$. For transaction t_1, $RS(t_1) = \{a,c\}$, $WS(t_1) = \{b\}$, and $write(t_1) = \{(b,5)\}$. Since only t_2 has a write operation on data item a and no transaction writes data item c, $\tau_w(a,c) = \{t_2\}$. □

We next introduce the notion of "state" which is a possible state the transaction might have seen. The state of a transaction is an abstract notion and may never have been physically realized in a schedule.

Definition 6 *Let S be a schedule, t be a transaction in S, $d \subseteq D$ and DS_1 be a database state such that $legal(DS_1, S)$. The state seen by t with respect to d exists if $(S^{\tau'})^d$ is serializable, where $\tau_w(d,S) \subseteq \tau'$. Without loss of generality let $t_1, t_2, \ldots, t_n$ be a serialization order of transactions in $(S^{\tau'})^d$. The state of an arbitrary transaction t_i, $i = 1,2,\ldots,n$ is defined recursively as follows.*

$$state(t_i, d, S, DS_1) = \begin{cases} DS_1^d & \text{if } i = 1 \\ state(t_{i-1}, d - WS(t_{i-1}), S, DS_1) \cup write(t_{i-1}^d), & \text{if } i > 1 \end{cases}$$

$state(t_i, d, S, DS_1)$ is the state of the database with respect to data items in d as seen by t_i. The state of a transaction depends on the initial state and the serialization order chosen and thus, may not be unique. Note that, $read(t_i^d) \subseteq state(t_i, d, S, DS_1)$.

Example 11 Consider the following schedule resulting from the execution of transaction programs tp_1 and tp_2 from database state $DS = \{(a,10),(b,5),(c,15)\}$.

$$S: \quad r_1(a,10) \quad r_2(a,10) \quad w_1(b,20) \quad w_2(c,25)$$

S is serializable with serialization order t_1, t_2 or t_2, t_1. With serialization order t_1, t_2,

$$state(t_2, \{a,b,c\}, S, DS) = \{(a,10),(b,20),(c,15)\}.$$

However, with serialization order t_2, t_1,

$$state(t_2, \{a,b,c\}, S, DS) = \{(a,10),(b,5),(c,15)\}. \ \square$$

The following lemmas relate the consistency of database states to the consistency of its subsets. In the proofs of the lemmas, $DS \models IC$ shall denote $I \models_{DS} IC$.

Lemma 2 *Let $IC = C_1 \wedge C_2 \wedge \cdots \wedge C_n$, where $C_1, C_2, \ldots, C_n$ are defined over data items in $d_1, d_2, \ldots, d_n$, where $d_i \cap d_j = \emptyset$ for all $i \neq j$. Let $d_i' \subseteq d_i$ and DS be a database state. $DS^{d_i'}$, for all i, $i = 1,2,\ldots,n$ is consistent iff $\bigcup_{i=1}^{n} DS^{d_i'}$ is consistent*[5].

[5] The $\cup$ operator is similar to the one traditionally defined for sets, except that $A \cup B$ is undefined if $(d', v_1') \in A$, $(d', v_2') \in B$ and $v_1' \neq v_2'$.

-Appendix A-

We shall, in this appendix, prove Theorems 1 to 5 developed in the paper. The proof of Theorem 1 is quite trivial and is developed next.

Proof of Theorem 1 (Trivial Model): As a result of the model, there is no edge in the serialization graph of S between nodes corresponding to transactions t_1, t_2 such that $t_1 \in \tau_G$ and $t_2 \in \tau_L$. Since S^{τ_G} is serializable, serialization graph of S^{τ_G} is acyclic. Also, S^{D_i} is serializable and local transactions at LDB_i do not conflict with local transactions at LDB_j, $i \neq j$. As a result, serialization graph of S^{τ_L} is acyclic. Thus, the serialization graph of S is acyclic and S is serializable. Hence proved. □

In order to prove the remainder of the theorems we need to develop the following notation. The developed notation is used in Appendix B as well.

Let seq be a sequence of operations. $RS(seq)$ denotes the set of data items read by operations in seq.

$$RS(seq) = \{y : o \in seq \wedge y = entity(o) \wedge action(o) = r\}$$

$WS(seq)$ denotes the set of data items written by operations in seq.

$$WS(seq) = \{y : o \in seq \wedge y = entity(o) \wedge action(o) = w\}$$

$write(seq)$ denotes the effect on the database as a result of the write operations in seq.

$$write(seq) = \{(y, z) : o \in seq \wedge y = entity(o) \wedge z = value(o) \wedge action(o) = w\}$$

We denote by $\tau_w(d, S)$ the set of transactions in a schedule S that have at least one write operation on some data item in $d \subseteq D$. Formally,

$$\tau_w(d, S) = \{t \in S : (WS(t) \cap d) \neq \emptyset\}$$

Example 10 Consider the schedule,

$$S: \quad r_2(d, 10) \quad r_1(a, 0) \quad w_2(a, 10) \quad r_1(c, 5) \quad w_1(b, 5)$$

resulting from the execution of transaction programs

$$tp_1: \quad \textbf{if}(a \geq 0) \quad \textbf{then} \quad b := c \qquad\qquad tp_2: \quad a := d$$
$$\textbf{else} \quad c := d$$

[5] Y. Breitbart, A. Silberschatz and G. Thompson, "Reliable Transaction Management in a Multidatabase System," Proceedings of ACM SIGMOD Conference, 1990.

[6] W. Du and A. Elmagarmid, "Quasi Serializability: A Correctness Criterion for Global Database Consistency in Interbase," Proceedings of the International Conference on Very Large Databases (VLDB), 1989.

[7] A. Elmagarmid and W. Du, "A Paradigm for Concurrency Control in Heterogeneous Distributed Database Systems," IEEE International Conference of Data Engineering, February, 1990.

[8] H. F. Korth, W. Kim and F. Bancilhon, "On Long-Duration CAD Transactions," Information Sciences 46, 1988.

[9] H. F. Korth and G. D. Speegle, "Formal Model of Correctness without Serializability," Proceedings of ACM SIGMOD Conference, 1988.

[10] S. Mehrotra, R. Rastogi, H. F. Korth and Avi Silberschatz, "On Correctness of Non-Serializable Executions," Submitted for Publication.

[11] C. Papadimitriou, "The Theory of Database Concurrency Control," Computer Science Press, 1986.

[12] D. Georgakopoulos and M. Rusinkiewicz, " On Serializability of Multidatabase Transaction through Forced Local Conflicts," Technical Report, University of Houston.

[13] K. Salem and H. Garcia-Molina, "Altruistic Locking," University of Maryland, Technical Report, UMIACS-TR-90-104, CS-TR-2515, 1990.

cases, that non-serializable executions preserve database consistency. We identified models for several of these cases, each of which involves different restrictions on a transaction's read and write operations. The models provide a range of options to the designer of an HDBMS. We have characterized the relative power of our models, both in terms of concurrency and restrictions imposed on transactions.

Most of the previous work for showing that non-serializable executions preserve database consistency resort to informal reasoning. This is mainly due to the limitation of the transaction models being currently used. For the purpose of dealing with non-serializable executions, we develop a transaction model which differs from other standard transaction models. Operations, in our transaction model, have values associated with them in addition to action and entity attributes. In addition, we formally define database consistency in terms of integrity constraints which are quantifier-free formulae over a first-order language.

Our approach is only the first step in relaxing the serializability requirement in an HDBMS environment. Some of the restrictions imposed by us on the model in order to preserve database consistency may be too severe for certain HDBMS applications. These restrictions could be relaxed by exploiting the exact nature of integrity constraints (e.g., replicated data). Further, fault tolerant algorithms for transaction management in HDBMS applications with these new correctness criteria still need to be developed.

References

[1] K. R. Apt, "Introduction to Logic Programming," Handbook of Theoretical Computer Science, (J. van Leeuwen, Managing Editor), North Holland.

[2] P. Bernstein, V. Hadzilacos and N. Goodman, "Concurrency Control and Recovery in Database Systems," Addison- Wesley Publishing Co., 1987.

[3] Y. Breitbart, "Multidatabase Interoperability," SIGMOD Record, 1990.

[4] Y. Breitbart and A. Silberschatz, "Multidatabase Update Issues," Proceedings of ACM SIGMOD Conference, 1988.

$$
\begin{array}{lll}
tp_1: & b_2 := 1; & \\
 & \textbf{if}(a \neq 1) \quad \textbf{then} & b_1 := 1 \\
 & & \\
tp_2: & b_2 := 0; & \\
 & \textbf{if}(c = 1) \quad \textbf{then} & \textbf{begin} \\
 & & \quad a := 0; \\
 & & \quad c := 0; \\
 & & \quad b_1 := 0 \\
 & & \textbf{end}; \\
 & \textbf{else} & b_1 := 0
\end{array}
$$

Let L be a local transaction program executing at site s_1.

$$
\begin{array}{ll}
L: & c := 0; \\
 & a := 0
\end{array}
$$

Transaction programs tp_1 and tp_2 have no value dependencies. Consider the following schedule resulting from the execution of tp_1, tp_2 and L from the database state $\{(a,1),(b_1,1),(c,1),(b_2,1)\}$.

$$
\begin{array}{llllll}
S_1: & r_1(a,1) & w_L(c,0) & w_L(a,0) & r_2(c,0) & w_2(b_1,0) \\
S_2: & w_2(b_2,0) & w_1(b_2,1) & & &
\end{array}
$$

The final global database state resulting from the above schedule is $\{(a,0),(b_1,0),(c,0),(b_2,1)\}$, which is inconsistent. □

8 Conclusion

In this paper, we adopt a weaker correctness criterion than serializability for HDBMS applications, which we refer to as two-level serializability (2LSR). The motivation for abandoning serializability as the correctness criterion is the low degree of concurrency that results from protocols for ensuring serializability in HDBMSs. two-level serializability only requires the projection of the global schedule on the set of global transactions to be serializable and each of the local schedules to be serializable. As a result, protocols for ensuring that schedules are 2LSR have the advantages of being simple, allowing a high degree of concurrency, and not violating the local autonomy of sites.

However, 2LSR schedules preserve database consistency only in certain restricted HDBMS applications. In many HDBMS applications, the data items for which intersite integrity constraints are introduced due to the integration of DBMSs are known (e.g., replicated data). We capture this knowledge in our HDBMS model by partitioning data items into two disjoint sets-global and local data items. A data item is a global data item if there is an integrity constraint between it and a data item at a different site. This knowledge allows us to prove, in certain

In [6], the authors claim that QSR as a correctness criterion in an HDBMS environment, requires that:

1. there can be no integrity constraints between data items at different sites (except equality constraints which model replicated data).
2. transaction programs must not have value dependencies.
3. local transactions must not be permitted to write replicated data.

QSR schedules can be shown to preserve database consistency within the framework that we have developed for studying transaction management in HDBMSs. To illustrate this, suppose that we further restrict the $G_{rw}L_r$ model as follows:

1. global data consists only of replicated data across sites.
2. no integrity constraints are present between local and global data (presence of such integrity constraints would result in inter-site integrity constraints).
3. transaction programs do not have value dependencies.

Using Theorem 5, the result that QSR schedules preserve database consistency under restrictions mentioned in [6] directly follows.

Theorem 5, however, states a more general result, in that restriction 1 mentioned above is not required, and schedules are required to be 2LSR, not QSR. Further, if the GTM follows the 2LPL protocol, then the requirement that no value dependencies be present between global subtransactions can be relaxed as shown in Theorem 6.

QSR schedules may not preserve database consistency if transaction programs have value dependencies as is demonstrated by Example 8. The following example illustrates that the presence of integrity constraints between replicated data and non-replicated data (this would result in inter-site integrity constraints besides replication) could result in the database consistency constraint being violated.

Example 9 Consider an HDBMS consisting of two sites s_1 and s_2. Let $D_1 = \{a, c, b_1\}$, $D_2 = \{b_2\}$, $LD = \{a, c\}$, and $GD = \{b_1, b_2\}$. Let $IC = (a = 1 \rightarrow b_1 = 1) \wedge b_1 = b_2 \wedge a = c$. Note that there is an implicit inter-site integrity constraint $(a = 1 \rightarrow b_2 = 1)$. Consider the following two global transaction programs.

$$L_1: \quad a := 1; \quad \textbf{if}(e > 0) \quad \textbf{then} \quad b := 1$$

$$L_2: \quad f := c$$

Consider the local schedules at sites s_1 and s_2 resulting from the execution of tp_1, tp_2, L_1 and L_2 from the database state $\{(a,-1),(b,-1),(c,1),(e,1),(f,1)\}$.

$S_1: \quad w_{L_1}(a,1) \quad r_1(a,1) \quad r_1(b,-1) \quad w_2(e,-1) \quad r_{L_1}(e,-1) \quad w_2(ticket_1) \quad w_1(ticket_1)$
$S_2: \quad w_1(c,-1) \quad r_{L_2}(c,-1) \quad w_{L_2}(f,-1) \quad r_2(f,-1)$

The final state resulting from the execution of the above schedule is $\{(a,1),(b,-1),(c,-1),(e,-1),(f,-$ which is inconsistent. □

Even if the GTM protocol were to be modified as follows: global transactions do not release a global lock unless all its operations at the local sites have completed, the above schedule could still be generated and database consistency could be violated unless the global subtransactions do a write on $ticket_i$ as their first operation. However, if local transaction programs have fixed-structure, the write on $ticket_i$ need not precede all other operations of the subtransaction, if database consistency is to be preserved (the proof can be found in Appendix B).

7 Related Work

Most of the work on transaction management in HDBMSs has been concentrated on ensuring global serializability and has not addressed the issue of non-serializable executions that preserve database consistency. An exception is the work by Du and Elmagarmid in which the notion of *quasi serializability (QSR)* was introduced [6]. In this section we discuss the relation of our work to QSR.

Definition 5 *A schedule S is* quasi serial *if and only if local schedules are serializable and there is a total order of global transactions such that for any two global transactions t_i and t_j if t_i precedes t_j in the total order, then all t_i operations precede all t_j operations at each and every local site. A schedule S is QSR if it is conflict equivalent to a quasi serial schedule.* □

Since in every QSR schedule, S^{τ_G} is serializable, QSR schedules are a subset of 2LSR schedules. Further, the schedule in Example 2 is a 2LSR schedule which is not QSR. Thus, QSR schedules are a proper subset of 2LSR schedules.

schedule, t_1 releases a global lock before t_2 releases a global lock, t_2 releases a global lock before t_3 releases a global lock, ..., t_n releases a global lock before t_1 releases a global lock. Thus, it follows that t_1 releases a global lock before t_n releases a global lock, which in turn releases a global lock before t_1 releases a global lock. Contradiction. Thus, S is serializable. □

In the presence of 2PL sites G2LPL schedules may not be serializable as is demonstrated by Example 7. However, the resulting schedules are strongly correct if there are no integrity constraints between local and global data items, and global transactions have fixed-structure.

Theorem 8 *Let S be a G2LPL schedule in the $G_{rw}L_r$ model. If there are no integrity constraints between the local and global data items, and the global transaction programs have fixed-structure, then S is strongly correct.* □

The importance of Theorem 6, and Theorem 8 is the implication that no restrictions need to be imposed on local transaction programs. Thus, the local autonomy of sites is not violated. However, it is necessary for the global transaction programs to have fixed-structure, since otherwise the hypothesis of Corollary 2, Theorem 6 and Theorem 8 do not hold. This is illustrated by Example 3 where the schedule is a 2LPL schedule, but since transaction program tp_1 does not have fixed-structure, database consistency is not preserved. Note that Theorem 6 is an instance of Theorem 8 when every LTM follows 2PL. Hence in the appendix we only prove Theorem 8.

Since local transactions programs do not have fixed-structure, it is essential the the write operation on $ticket_i$ by a global subtransaction precede all other operations of the subtransaction; else, database consistency may be violated as is illustrated by the following example.

Example 8 Consider an HDBMS consisting of a non-2PL site s_1 and and a 2PL site s_2. Let $D_1 = \{a, b, e, ticket_1\}$, $D_2 = \{c, f\}$, $LD = \{a, b, c, e, f\}$, and $GD = \{\}$. Let $IC = (a > 0 \rightarrow b > 0) \wedge c > 0 \wedge e > 0 \wedge f > 0$. Thus, no integrity constraints are present between local and global data items. Consider the following two global transaction programs.

$$tp_1: \quad \textbf{if}(a > 0) \quad \textbf{then} \quad c := b \qquad \textbf{else} \quad c := 1$$

$$tp_2: \quad e := f$$

Let L_1 be a local transaction program executing at site s_1 and L_2 be a local transaction program executing at site s_2.

- Every global transaction obtains the global lock corresponding to a data item before accessing the data item.
- A global transaction, once it releases a global lock, does not obtain any more global locks.
- A global transaction does not obtain a global lock that is held by any other global transaction.
- A global transaction does not release a global lock unless all local locks corresponding to global locks held by the global transaction for data items at the 2PL sites have been obtained. In addition, a global lock corresponding to a data item accessed by the global transaction at a non-2PL site is released by a global transaction only after the operation on the data item is completed (at the local sites).
- Every global transaction with operations at a non-2PL site s_i, does a $write(ticket_i)$ before executing any of its operations at s_i.

Thus, if every LTM follows the 2PL protocol, then every G2LPL schedule is 2LPL. However, if some of the LTMs do not follow the 2PL locking protocol, then G2LPL schedules may not be 2LPL, as is illustrated by the following example.

Example 7 Consider an HDBMS consisting of a non-2PL site s_1 and a 2PL site s_2. Let the local schedules at the sites be

$$\begin{array}{lllll} S_1: & w_2(ticket_1) & w_1(ticket_1) & r_{L_1}(a) & w_2(a) \\ S_2: & w_1(b) & r_{L_2}(b) & w_{L_2}(c) & r_2(c) \end{array}$$

The schedule in the above example is a G2LPL schedule which is not 2LPL since S_1 is not a 2PL schedule. □

As with 2LPL schedules, G2LPL schedules are 2LSR. Furthermore, if no LTM follows 2PL (or we do not know about the protocol followed by the LTMs) G2LPL schedules are serializable.

Theorem 7 *Every G2LPL schedule consisting of only non-2PL sites is serializable.*

Proof: Suppose a G2LPL schedule S consisting of only non-2PL sites is non-serializable. Thus, there is a cycle in the serialization graph of schedule S. Since every local schedule is serializable, the cycle must contain at least two global transactions. We consider the cycle consisting of just global transactions (as any two global transaction have a conflict at every site they execute in common). Let $t_1, t_2, \ldots, t_n, t_1$ be a cycle in S consisting of only global transactions such that t_1 conflicts with t_2, t_2 conflicts with t_3, ..., t_n conflicts with t_1. Since S is a G2LPL

- A global transaction does not release a global lock until all local locks corresponding to global locks held by the global transaction have been obtained[4].

We refer to the schedules obtained as a result of following the above scheme as *two-level two-phase locking (2LPL)* schedules. Every 2LPL schedule is a 2LSR schedule, but not vice-versa, as is illustrated by the schedule in Example 8 which is 2LSR but not 2LPL (since S_1 is not a 2PL schedule). In the L_r and $G_{rw}L_r$ models, for 2LSR schedules to preserve database consistency, transaction programs are required to be LDP. However, if schedules are 2LPL, this restriction can be relaxed, and database consistency is preserved in these models even if transaction programs have value dependencies.

Theorem 6 *Let S be a 2LPL schedule in the $G_{rw}L_r$ model. If there are no integrity constraints between the local and global data items, and the global transaction programs have fixed-structure, then S is strongly correct.* □

As the $G_{rw}L_r$ model is a generalization of the L_r model, the following corollary trivially follows.

Corollary 2 *Let S be a 2LPL schedule in the L_r model. If there are no integrity constraints between the local and global data items, and the global transaction programs have fixed-structure, then S is strongly correct.* □

It is an open problem, whether a 2LPL schedule in the L_r model is strongly correct if integrity constraints between local and global data items are permitted, and global transactions have fixed-structure. In order for schedules to be 2LPL, every LTM is required to follow the 2PL locking protocol. We now define *generalized two-level two-phase (G2LPL)* schedules, which do not require that each LTM follow the 2PL locking protocol. We refer to sites which follow the 2PL locking protocol as the *2PL sites* and the sites which do not follow 2PL (or we have no information about the concurrency control protocol at the site) as the *non-2PL sites*.

G2LPL schedules result from a scheme similar to the one resulting in 2LPL schedules. In addition, every non-2PL site s_i is assumed to contain a data item $ticket_i$. The following GTM protocol results in G2LPL schedules.

[4]If the LTM interface does not provide for the submission of explicit lock operations, then the GTM can utilize the acknowledgements for operations to determine if local locks have been obtained.

violated if transaction programs are not LDP. The following theorem states conditions under which schedules preserve database consistency in the $G_{rw}L_r$ model.

Theorem 5 *Let S be a 2LSR schedule in the $G_{rw}L_r$ model. If all transaction programs are LDP, and no integrity constraints are present between local and global data items, then S is strongly correct.* □

6 Concurrency Control Protocols

In order to ensure that a schedule S is two-level serializable, the GTM protocol only needs to ensure that S^{τ_G} is serializable, since the LTM at each site s_i ensures the serializability of S^{D_i}. Thus, in an HDBMS, two-level serializability can easily be ensured, since the global transactions execute under the control of the GTM. For example, the GTM could follow a protocol very similar to 2PL, in which the GTM maintains locks for every data item accessed by global transactions (we refer to them as *global locks*). In addition, in order to ensure that S^{τ_G} is serializable,

- Every global transaction obtains the global lock for a data item before accessing the data item.
- A global transaction, once it releases a global lock, does not obtain any more global locks.
- A global transaction does not obtain a global lock that is held by any other global transaction.
- A global lock is held by the global transaction at least until the completion of the operation (at the local site) on the data item for which the lock was obtained.

Let us consider a variant of the above simple scheme in which each LTM follows the 2PL protocol. We refer to locks maintained by the LTMs for the data items at the local sites as *local locks*. Thus, corresponding to a data item accessed by global transactions, both the GTM and LTM maintain locks. The GTM protocol is as follows.

- Every global transaction obtains the global lock for a data item before accessing the data item.
- A global transaction, once it releases a global lock, does not obtain any more global locks.
- A global transaction does not obtain a global lock that is held by any other global transaction.

Definition 4 *Transaction program tp has fixed-structure if for all pairs (DS_1, DS_2) of database states, $struct(t_1) = struct(t_2)$, where t_1 and t_2 are the transactions resulting from the execution of tp from DS_1 and DS_2 respectively.* □

Example 6 Consider the following transaction programs tp_1 and tp_2. While tp_1 is a fixed-structure transaction program, tp_2 is not:

$$tp_1: \ \textbf{if}(x > 5) \ \textbf{then} \ y := 3 \ \textbf{else} \ y := 5 \qquad tp_2: \ \textbf{if}(x > 5) \ \textbf{then} \ y := 3 \ \textbf{else} \ z := 5$$

□

If we restrict all transaction programs to have fixed-structure, then 2LSR schedules preserve database consistency.

Theorem 4 *Let S be a 2LSR schedule in the G_{rw} model resulting from the execution of fixed-structure transaction programs. If no integrity constraints are present between local and global data items, then S is strongly correct.* □

Note that even local transaction programs must have fixed-structure, if database consistency is to be preserved. In Example 8, only the local transaction program did not have fixed-structure and that resulted in the database inconsistency. This may not be a very practical assumption since local transaction programs are pre-existing and may not satisfy the above restriction. Another way of ensuring that 2LSR schedules are strongly correct is to restrict transaction programs to be LDP. In the next subsection, it is shown that, under this restriction, 2LSR schedules are strongly correct for the $G_{rw}L_r$ model, which is more general than the G_{rw} model.

5.5 The Global Read-Write and Local Read ($G_{rw}L_r$) Model

Consider a model of HDBMS application with the following restriction on transactions.

- local transactions read and write local data items and also read global data items, and
- global transactions read and write global and local data items.

The $G_{rw}L_r$ model is more general than all the models considered so far. Since the $G_{rw}L_r$ model is more general than the G_r model, presence of integrity constraints between local and global data items may result in the violation of database consistency as was illustrated in Example 2. Similarly, as shown in Example 3, in the L_r model, database consistency may be

5.4 The Global Read-Write (G_{rw}) Model

Consider a model of HDBMS applications with the following restrictions on transactions:

- local transactions only read and write local data items, and
- global transactions, in addition to reading and writing global data items, also read and write local data items.

The G_{rw} model is more general than the G_r model. Thus, as shown in Example 2, in the G_{rw} model, 2LSR schedules may not preserve database consistency if integrity constraints are present between local and global data items. However, in contrast to the G_r model, absence of integrity constraints between local and global data items does not ensure database consistency as is demonstrated by the following example.

Example 5 Consider an HDBMS consisting of two sites s_1 and s_2. Let $D_1 = \{a, b, e\}$, $D_2 = \{c\}$, $LD = \{a, b, c, e\}$, and $GD = \{\}$. Let $IC = (a > 0 \rightarrow b > 0) \wedge c > 0 \wedge e > 0$. Thus, no integrity constraints are present between local and global data items. Consider the following two global transaction programs.

$$tp_1: \quad \textbf{if}(a > 0) \quad \textbf{then} \quad c := b \qquad \textbf{else} \quad c := 1$$

$$tp_2: \quad e := c$$

Let L be a local transaction program executing at site s_1.

$$L: \quad a := 1; \qquad \textbf{if}(e > 0) \quad \textbf{then} \quad b := 1$$

Consider the local schedules at sites s_1 and s_2 resulting from the execution of tp_1, tp_2 and L from the database state $\{(a, -1), (b, -1), (c, 1), (e, 1)\}$.

$$S_1: \quad w_L(a, 1) \quad r_1(a, 1) \quad r_1(b, -1) \quad w_2(e, -1) \quad r_L(e, -1)$$
$$S_2: \quad w_1(c, -1) \quad r_2(c, -1)$$

The final state resulting from the execution of the above schedule is $\{(a, 1), (b, -1), (c, -1), (e, -1)\}$, which is inconsistent. □

In Example 8, database inconsistency results from the fact that the structure of transaction program L changes based on the value returned by the read operation on data item e. To avoid such inconsistencies, we need to restrict the transaction programs to be *fixed-structured*. A transaction program is *fixed-structured* if its execution from every database state result in transactions with the same structure.

Consider the transaction program tp, which has a single assignment statement $a := b$, where a and b are data items. If a and b are at different sites, then tp has a value dependency since the write on a depends on the read operation which returns the value of b. However, if a and b were at the same site, tp would have no value dependency.

The following Lemma relates the notions of value dependencies and LDP.

Lemma 1 *If a transaction program tp has no value dependencies, then tp is LDP.* □

Proof: Let DS_1 be an arbitrary database state such that $DS_1^{D_i}$ is consistent, for an arbitrary i, $i = 1, 2, \ldots, m$. Let $\{DS_1\}tp\{DS_2\}$. In order to show that tp is LDP, we need to show that $DS_2^{D_i}$ is consistent. Since $DS_1^{D_i}$ is consistent, by the definition of a consistent state, there exists a consistent database state DS_3 such that $DS_3^{D_i} = DS_1^{D_i}$. Let $\{DS_3\}tp\{DS_4\}$. Let t_1, t_2 be transactions resulting from the execution of tp from DS_1, DS_3 respectively. Since it is given that tp has no value dependencies, $t_1^{D_i} = t_2^{D_i}$. Thus, $DS_4^{D_i} = DS_2^{D_i}$. Since tp preserves database consistency, DS_4 is consistent. As $DS_4^{D_i}$ is consistent, $DS_2^{D_i}$ is consistent. Hence proved. □

Since local transaction programs execute at a single site, they have no value dependencies. The following corollary trivially follows.

Corollary 1 *Every local transaction program is LDP.* □

In the L_r model, 2LSR schedules may not preserve database consistency as was shown in Example 3. However, if we restrict the transaction programs to be LDP, then 2LSR schedules can be shown to be strongly correct.

Theorem 3 *Let S be a 2LSR schedule in the L_r model. If all transaction programs are LDP, then S is strongly correct.* □

As a transaction program with no value dependencies is LDP, Theorem 3 holds if transaction programs have no value dependencies. However, if transaction programs have value dependencies, then schedules may not preserve database consistency as illustrated in Example 4. It must be noted that Theorem 3 holds even if integrity constraints are present between local and global data items.

Another example of 2LSR schedules not preserving database consistency in the L_r model is given below.

Example 4 Consider an HDBMS consisting of two sites s_1 and s_2. Let $D_1 = \{a, e, b\}$, $D_2 = \{c\}$, $LD = \{b\}$ and $GD = \{a, c, e\}$. Let $IC = (c > 0 \rightarrow a > 0) \wedge (e > 0 \rightarrow c > 0) \wedge (e > 0 \rightarrow a > 0) \wedge b > 0$. Note that no integrity constraints are present between local and global data items. Consider the following two global transaction programs.

$$\begin{array}{lll} tp_1: & a := 1; & \\ & c := 1 & \end{array}$$

$$\begin{array}{lllll} tp_2: & \textbf{if}(c > 0) & \textbf{then} & e := 1 \\ & & \textbf{else} & e := -1 \end{array}$$

Let L be a local transaction program executing at site s_1.

$$\begin{array}{lll} L: & temp := a; & \\ & \textbf{if}(e \leq 0) \quad \textbf{then} & temp := 1; \\ & b := temp & \end{array}$$

Consider the local schedules at sites s_1 and s_2 resulting from the execution of tp_1, tp_2 and L from the database state $\{(a, -1), (e, -1), (c, -1), (b, 1)\}$.

$$\begin{array}{llllll} s_1: & r_L(a, -1) & w_1(a, 1) & w_2(e, 1) & r_L(e, 1) & w_L(b, -1) \\ s_2: & w_1(c, 1) & r_2(c, 1) & & & \end{array}$$

The final state resulting from the execution of the above schedule is $\{(a, 1), (e, 1), (c, 1), (b, -1)\}$, which is inconsistent. □

In Example 4, the local transaction program L reads inconsistent data values. The reason for the reads being inconsistent is that the value written by the global transaction program tp_2 for the data item e depended on the value of data item c, which is located at another site. As the global transaction programs can access data at multiple sites, the "behavior" of a global transaction at a site may depend on the database state "seen" by it at a different site. One way to remove such inconsistent reads is by restricting the global transaction programs.

A restriction similar to LDP, absence of *value dependency*, was imposed on transaction programs in [7]. Informally, a global transaction program has value dependencies if the operations performed by it at one site depend on the operations it has performed at another site.

Definition 3 *Transaction program tp has no value dependency if for all pairs (DS_1, DS_2) of database states and for all i, $i = 1, 2, \ldots, m$, if $DS_1^{D_i} = DS_2^{D_i}$, then $t_1^{D_i} = t_2^{D_i}$, where t_1 and t_2 are transactions resulting from the execution of tp from DS_1 and DS_2 respectively.* □

5.3 The Local Read (L_r) Model

Consider a model of HDBMS applications with the following restrictions on transactions:

- local transactions read and write local data items and also read global data items, and
- global transactions only read and write global data items.

As in the G_r model 2LSR schedules may not always preserve database consistency in the L_r model as the following example illustrates.

Example 3 Consider an HDBMS consisting of two sites s_1 and s_2. Let $D_1 = \{b, c, e\}$, $D_2 = \{a\}$, $LD = \{e\}$ and $GD = \{a, b, c\}$. Let $IC = (a > 0 \rightarrow c > 0) \wedge (b > 0 \rightarrow c > 0) \wedge e > 0$. Thus, no integrity constraints are present between local and global data. Consider the following two global transaction programs.

$$\begin{array}{lll} tp_1: & b := 1; & \\ & \textbf{if}(a \leq 0) \quad \textbf{then} & c := 1 \end{array}$$

$$\begin{array}{lll} tp_2: & c := 1; & \\ & a := 1 & \end{array}$$

Let L be a local transaction program executing at site s_1.

$$\begin{array}{lll} L: & \textbf{if}(b > 0) \quad \textbf{then} & e := c \\ & \textbf{else} & e := 1 \end{array}$$

Consider the local schedules at sites s_1 and s_2 resulting from the execution of tp_1, tp_2 and L from the database state $\{(a, -1), (b, -1), (c, -1), (e, 1)\}$.

$$\begin{array}{lllllll} S_1: & w_1(b,1) & r_L(b,1) & r_L(c,-1) & w_2(c,1) & w_L(e,-1) \\ S_2: & w_2(a,1) & r_1(a,1) & & & \end{array}$$

The final state resulting from the execution of the above schedule is $\{(a, 1), (b, 1), (c, 1), (e, -1)\}$, which is inconsistent. □

In Example 3, the local transaction program L reads inconsistent data values. The reason for the reads being inconsistent is that tp_1 leaves the database at site s_1 in an inconsistent state, which is then "seen" by L. This can be overcome by requiring the execution of global transaction programs from a consistent local database state to leave the local database in a consistent state (other local database states might be inconsistent). This is stated more formally below.

Definition 2 *Transaction program tp is* Local Database Preserving (LDP) *if for all database states DS_1 and for all i, $i = 1, 2, \ldots, m$, if $DS_1^{D_i}$ is consistent, and $\{DS_1\}tp\{DS_2\}$, then $DS_2^{D_i}$ is consistent.* □

- local transactions only read and write local data items, and
- global transactions in addition to reading and writing global data items, also read local data items.

In the G_r model 2LSR schedules may not preserve database consistency as is illustrated by the following example.

Example 2 Consider an HDBMS consisting of two sites s_1 and s_2. Let $D_1 = \{a, b, e\}$, $D_2 = \{c\}$, $LD = \{a\}$ and $GD = \{b, c, e\}$. Let $IC = (a > 0 \rightarrow b > 0) \wedge (c > 0 \rightarrow (b > 0 \vee e > 0))$. Note that $(a > 0 \rightarrow b > 0)$ is an integrity constraint between local data item a and global data item b. Consider the following global transaction programs.

$$\begin{array}{lllll} tp_1: & \textbf{if}(a \leq 0) & \textbf{then} & e := 1 \\ & & \textbf{else} & e := -1; \\ & c := 1 & & \end{array}$$

$$\begin{array}{lllll} tp_2: & \textbf{if}(a \leq 0) & \textbf{then} & b := -1 \\ & & \textbf{else} & b := 1; \\ & c := -1 & & \end{array}$$

Let L be a local transaction program executing at site s_1.

$$L: \quad a := -1$$

Consider the local schedules at sites s_1 and s_2 resulting from the execution of tp_1, tp_2 and L from the database state $\{(a, 1), (c, 1), (b, 1), (e, 1)\}$.

$$\begin{array}{llllll} S_1: & r_1(a, 1) & w_1(e, -1) & w_L(a, -1) & r_2(a, -1) & w_2(b, -1) \\ S_2: & w_2(c, -1) & w_1(c, 1) & & & \end{array}$$

The final global database state resulting from the above schedule is $\{(a, -1), (c, 1), (b, -1), (e, -1)\}$, which is inconsistent. □

In Example 2 above, note that there is an integrity constraint between a which is a local data item and b which is a global data item. If we further restrict the model and disallow such integrity constraints between local and global data items, 2LSR schedules can be proved to be strongly correct.

Theorem 2 *Let S be a 2LSR schedule in the G_r model. If no integrity constraints are present between local and global data items, then S is strongly correct.* □

In an HDBMS, pre-existing local applications can be assumed to preserve local database consistency constraints which were present prior to integration. However, they are unaware of the newly introduced integrity constraints. As a result, if local transactions were to write global data items, database consistency may not be preserved. For example, the resulting database state may be inconsistent if local transactions wrote replicated data items. Local transactions, thus, do not write global data items. In the remainder of the paper, we assume that all transaction programs, when executed in isolation, preserve global database consistency. We also assume that each LTM follows a concurrency control protocol that ensures serializability at local sites.

5 Two-Level Serializability

A schedule S is *two-level serializable (2LSR)* if its projection on the set of global transactions is serializable and its projection on data items at each site is serializable. That is, a schedule $S = (\tau, \prec_S)$ is 2LSR if S^{τ_G} is serializable and S^{D_i}, $i = 1, 2, \ldots, m$ is serializable.

Every serializable schedule is 2LSR, but not vice versa. Unlike serializable schedules, 2LSR schedules may not preserve database consistency. In this section, we identify restrictions under which 2LSR schedules preserve database consistency in various models of HDBMS applications. These models are based on restricting the transactions' read and write operations of the various data items.

5.1 The Trivial Model

Consider a model of HDBMS applications with the following restrictions on the transactions:

- local transactions only read and write local data items, and
- global transactions only read and write global data items.

For this model we can establish the following theorem.

Theorem 1 *Let S be a 2LSR schedule in the trivial model. S is serializable and thus is strongly correct.* □

5.2 The Global Read (G_r) Model

Consider a model of HDBMS applications with the following restrictions on transactions:

- **Global transactions.** The set of transactions, denoted by τ_G, that may execute at more than one site. Global transactions are normally applications developed after the integration has been performed.

Each local database system has a *local transaction manager (LTM)* which is responsible for ensuring local database consistency. The *global transaction manager (GTM)*, built on top of the existing databases, is responsible for ensuring global database consistency. The GTM controls the execution of global transactions. We assume that the interface between the GTM and the LTMs provides for operations to be submitted by the GTM to the LTMs and the LTMs to acknowledge the completion of operations to the GTM.

A distinguishing feature of an HDBMS is the requirement that each local DBMS retains its *local autonomy*. In this paper, local autonomy is defined to mean:

- The local sites are free to follow any concurrency control protocol to ensure local database consistency.
- The LTMs do not communicate any information (e.g., conflict graph) relevant for concurrency control to the GTM.
- The local transactions execute outside the control of the GTM.
- Pre-existing local applications are able to execute without any changes being made to them.

Each DBMS defines certain local consistency constraints among the various local data items. The integration of the various DBMSs into an HDBMS results in the introduction of certain global inter-site constraints, which were not present prior to integration. We can thus partition the set of data items at a site, D_i, into *local* data items, LD_i, and *global* data items, GD_i, such that $LD_i \cap GD_i = \emptyset$, $D_i = LD_i \cup GD_i$, and if there is an integrity constraint between $d_1' \in D_i$ and $d_2' \in D_j$, $i \neq j$, then $d_1' \in GD_i$ and $d_2' \in GD_j$. The set of all the global data items, $GD = \bigcup_{i=1}^{m} GD_i$. Data items at different sites between which integrity constraints are introduced as a result of the integration are thus in GD. For example all replicated data items are global (We model replicated data items as distinct data items at different sites with an equality constraint between them).

3.3 Strong Correctness

In a database system, transaction programs, when executed in isolation, are assumed to preserve the integrity constraints of the database. The task of the concurrency control scheme is to ensure that schedules resulting from the concurrent execution of the transaction programs preserve database consistency. However, a concurrency control scheme which ensures that schedules preserve the database integrity constraints does not prevent transactions from "seeing" inconsistent database states. To overcome this, we define the notion of *strong correctness*, which requires that transactions in a schedule read consistent data values, in addition to the requirement that schedules preserve database integrity constraints.

Definition 1 *A schedule $S = (\tau, \prec_S)$ is strongly correct iff*

- *for all consistent database states DS_1, if $\{DS_1\}S\{DS_2\}$, then DS_2 is consistent, and*
- *for all transactions $t \in \tau$, read(t) is consistent.* □

Every serializable[2] schedule is strongly correct, but there are strongly correct schedules that are not serializable. The correctness criterion that we develop in this paper, two-level serializability, ensures schedules in certain HDBMS applications are strongly correct without requiring serializability.

4 The HDBMS Model

A heterogeneous distributed database consists of a set of autonomous pre-existing local databases $LDB_1, LDB_2, \ldots, LDB_m$ located at sites $s_1, s_2, \ldots, s_m$ respectively. Each LDB_i consists of a set of data items D_i. We denote the set of all the data items in the heterogeneous distributed database by D; thus $D = \bigcup_{i=1}^{m} D_i$. We assume that the local databases are disjoint; that is, $D_i \cap D_j = \emptyset$, $i \neq j$.

Integrity constraints are as defined in Section 2, and transactions and schedules[3] are as defined in Section 3. In our model, transactions are of two types:

- **Local transactions.** The set of transactions, denoted by τ_L, that execute at a single site. All pre-existing applications running at a local site before integration are local transactions.

[2] By serializability, in this paper, we refer to conflict serializability (CSR) [11].

[3] In distributed systems, schedules are generally defined to be a set of operations with a partial order defined on them. We consider schedules to be a set of operations with a total order defined on them. This total order can be assumed to be an arbitrary total order which is consistent with the original partial order.

Equality over the structure of sequences is defined similarly, except that values associated with operations are ignored. If two sequences are equal, then their structures are equal, but not vice versa.

For the remainder of the paper, we shall use t_i to denote the transaction resulting from the execution of the transaction program tp_i. Operations belonging to transaction t_i will be subscripted by i. Thus, a read operation on data item a belonging to transaction t_1 will be denoted by $r_1(a, v)$, where v is the value returned by the read.

Example 1 Consider the following transaction program:

$$tp_1: \quad \textbf{if}(a = 0) \quad \textbf{then} \quad b := 0 \quad \textbf{else} \quad c := 0$$

Execution of tp_1 from database state $DS_1 = \{(a, 0), (b, 5), (c, 3)\}$ results in the following transaction:

$$t_1: \quad r_1(a, 0) \quad w_1(b, 0)$$

where,

$$struct(t_1): \quad r_1(a) \quad w_1(b)$$

$$t_1^{\{b\}}: \quad w_1(b, 0)$$

$$read(t_1) = \quad \{(a, 0)\}$$

However, execution of tp_1 from database state $DS_3 = \{(a, 10), (b, 12), (c, 15)\}$ results in:

$$t_1': \quad r_1'(a, 10) \quad w_1'(c, 0)$$

Also, execution of tp_1 from DS_1 results in a database state DS_2, where $DS_2 = \{(a, 0), (b, 0), (c, 3)\}$. Thus, $\{DS_1\}tp_1\{DS_2\}$. □

3.2 Schedules

A schedule consists of a sequence of operations resulting from the concurrent execution of a set of transaction programs. A schedule $S = (\tau, \prec_S)$ is a finite set τ of transactions, together with a total order, $\prec_S$, on all operations of the transactions. Also, if for two operations o_1, o_2 in S and some transaction $t \in \tau$ we have $o_1 \prec_t o_2$, then $o_1 \prec_S o_2$. In order to develop properties of schedules we shall need to consider projections of schedules on sets of data items and transactions. Let $d \subseteq D$, and $\tau' \subseteq \tau$. We denote by S^d the projection of S on data items in d, and by $S^{\tau'}$ the projection of S on transactions in τ'.

addition to action and entity attributes. Since we relax the requirement of serializability as the correctness criterion, we need to deal with certain non-serializable executions. The value attribute helps us in proving that such non-serializable executions preserve database consistency.

We use the notation $\{DS_1\}\ tp\ \{DS_2\}$ to denote the fact that when transaction program tp executes from a database state DS_1, it results in a database state DS_2. Similar notation is used to denote execution of operations, transactions and schedules (the intended meaning will be clear from the context). Since operations have values associated with them, execution of operations is possible only from certain database states. A database state DS is *legal* with respect to operation o_i, denoted by $legal(DS, o_i)$, if it is possible to execute o_i from DS. Thus, $legal(DS, o_i)$ if

- either $action(o_i) = w$ or
- if $action(o_i) = r$, then $(entity(o_i), value(o_i)) \in DS$.

A database state DS is *legal* with respect to a sequence of operations $o_1 o_2 \ldots o_p$ if it is possible to execute $o_1 o_2 \ldots o_p$ from DS; that is, $legal(DS, o_1 o_2 \ldots o_p)$ if:

- $legal(DS, o_1)$, and
- if $p > 1$, then $legal(DS', o_2 \ldots o_p)$, where $\{DS\}\ o_1\ \{DS'\}$.

Execution of a sequence of operations $o_1 o_2 \ldots o_p$ from a database state which is not legal with respect to $o_1 o_2 \ldots o_p$ is undefined.

In order to discuss properties of transaction executions we introduce the following notation. Let $seq = o_1 o_2 \ldots o_p$ be a sequence of operations. The sequence seq has a structure associated with it denoted by $struct(seq)$, which is derived from seq by ignoring the values associated with the operations in seq. Thus every operation o_i in $struct(seq)$ is a 2-tuple $(action(o_i), entity(o_i))$. Let $d \subseteq D$. seq^d denotes the subsequence of seq consisting of all operations $o \in seq$ such that $entity(o) \in d$. $read(seq)$ denotes the database state "seen" as a result the read operations in seq. Formally,

$$read(seq) = \{(y, z) : o \in seq \wedge y = entity(o) \wedge z = value(o) \wedge action(o) = r\}$$

Two sequences of operations seq_1 and seq_2 are equal, denoted by $seq_1 = seq_2$, if the operations in the sequences are the same and the order of operations in both seq_1 and seq_2 is also the same.

to data items in the set d, where $d \subseteq D$. Thus, $DS^d = \{(d', v') : d' \in d \text{ and } (d', v') \in DS\}$. DS^d is consistent iff there exists a consistent database state DS_1 such that $DS_1^d = DS^d$. For example, consider a database consisting of data items a and b and $IC = (a = b)$. A database state $DS = \{(a,5),(b,6)\}$ is not consistent. However, $DS^{\{a\}} = \{(a,5)\}$ is consistent and $DS^{\{b\}} = \{(b,6)\}$ is consistent.

3 Transaction Model

In this section, we develop the transaction model, which deviates from the standard development to facilitate reasoning about non-serializable executions. We also develop *strong correctness* as a requirement on schedules. Strong correctness requires more than just preservation of database consistency, as we shall see in what follows.

3.1 Transactions

A transaction is a sequence of operations resulting from the execution of a *transaction program*. A transaction program is usually written in a high level programming language with assignments, loops, conditional statements and other complex control structures. Thus, execution of a transaction program starting at different database states may result in different transactions. This observation has serious implications when dealing with non-serializable executions as will become evident later in the paper.

Formally, a transaction $t = (O, \prec_t)$, where $O = \{o_1, o_2, \ldots, o_n\}$ is a set of operations and $\prec_t$ is a total order on O. An operation o_i is a 3-tuple $(action(o_i), entity(o_i), value(o_i))$. $action(o_i)$ denotes an operation type, which is either a read (r) or write (w) operation. $entity(o_i)$ is the data item on which the operation is performed. If the operation is a read operation, $value(o_i)$ is the value returned by the read operation for the data item read. For a write operation, $value(o_i)$ is the value assigned to the data item by the write operation. We assume, that for each transaction, a database item is read at most once and written at most once, and that no database item is read after it is written.

Our transaction definition differs from the way they are traditionally defined in the literature (see for example [2], [11]). We include, along with every operation, a value attribute, in

partitioning of data at each site is introduced in Section 4. In Section 5, we identify restrictions on transactions and integrity constraints in HDBMSs under which 2LSR schedules preserve global database consistency. In Section 6, we present a protocol which ensures that schedules are 2LSR. Concluding remarks are offered in Section 7.

2 Database Consistency

In the standard transaction model [11], a consistent database state is implicitly defined by assuming that each transaction, when executed in isolation, maps a consistent database state to another consistent database state. Correctness in case of concurrent execution is defined in terms of serializability. In order to develop a theory of non-serializable executions, we must explicitly define what a consistent database state is. We do this in terms of integrity constraints, which are discussed below.

A database consists of a countable set, D, of data items. For each data item $d' \in D$, $Dom(d')$ denotes the domain of d'. A *database state* maps every data item d' to a value v', where $v' \in Dom(d')$. Thus, a database state can be expressed as a set of ordered pairs of data items in D and their values, $DS = \{(d', v') : d' \in D \text{ and } v' \in Dom(d')\}$[1].

Integrity constraints (denoted by IC) in a database distinguish inconsistent database states from consistent ones. Traditionally, integrity constraints are defined as a subset of all the possible database states, and a database state is consistent if it belongs to that subset [11]. In our model, integrity constraints are quantifier-free formulae over a first order language consisting of:

- Numerical and string constants (e.g., 5, 100, 'Jim'),
- Functions over numeric and string constants (e.g., +, max),
- Comparison operators (e.g., >, =), and
- Set of variables (data items in D).

The terms and well-formed formulae are defined as in [1]. Let I be the standard interpretation for numerical and string constants, function symbols, and comparison operators. Since a database state maps data items (variables) to values it can be viewed as a *variable assignment* [1]. A database state DS is consistent iff $I \models_{DS} IC$. DS^d denotes the state of the database restricted

[1] DS has the property that if $(d', v'_1) \in DS$ and $(d', v'_2) \in DS$, then $v'_1 = v'_2$.

which they execute. Since the *global transaction manager (GTM)* has no knowledge about the local transactions and the indirect conflicts caused by them among the global transactions, the GTM cannot, without taking additional steps, determine the serialization order of transactions at a site. The steps taken by the GTM to ensure the above conditions are pessimistic; that is, it prevents non-serializable schedules from being generated by either suitably restricting the concurrent execution of global transactions (as in [4],[7]), or by aborting global transactions that could potentially result in non-serializable executions (as in [12]). Thus, it seems that any protocol for ensuring global serializability in an HDBMS will provide low concurrency.

One way of increasing concurrency is to relax the serializability requirement [3]. For example, the *quasi-serializability (QSR)* correctness criterion introduced in [6] preserves database consistency under the restrictions that no inter-site integrity constraints besides replication are present, and global transactions do not have *value dependencies* (that is, operations of a global transaction at a site are independent of its operations at other sites). The *altruistic locking* scheme presented in [13] is a simple protocol that ensures quasi-serializability.

In this paper, we follow on the idea of relaxing the correctness criterion for HDBMS applications to allow the inclusion of non-serializable executions. We develop a new correctness criterion, *two-level serializability (2LSR)*, which is more general than both serializability and quasi-serializability. We identify restrictions on transaction programs and integrity constraints in HDBMSs under which 2LSR schedules preserve database consistency. A simple protocol for ensuring two-level serializability in HDBMSs that does not violate the local autonomy of sites is also presented. It is shown that the restrictions imposed on global transactions in [6] (absence of value dependencies between global subtransactions) in order to preserve database consistency can be relaxed if the GTM follows a variant of the protocol.

The paper contains a number of theorems and lemmas, whose proofs are of considerable length and can be found in the appendix.

The remainder of the paper is organized as follows. In Section 2, we define database consistency in terms of integrity constraints. A transaction model suited to dealing with non-serializable schedules is developed in Section 3. A model for HDBMS applications based on the

Maintaining Database Consistency in Heterogeneous Distributed Database Systems*

Sharad Mehrotra
Rajeev Rastogi
Henry F. Korth
Abraham Silberschatz

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

Abstract

The concept of serializability has been the traditionally accepted notion of correctness in database systems. However, in a heterogeneous distributed database system (HDBMS) environment, ensuring serializability is a difficult task mainly due to the desire of preserving the local autonomy of the various participating local database systems. In this paper, we introduce a new correctness criterion for HDBMSs, *two-level serializability (2LSR)*. We identify restrictions under which 2LSR schedules preserve database consistency. Further, we present a simple protocol for ensuring schedules are 2LSR. This protocol is easily implementable and does not violate the local autonomy of sites.

1 Introduction

The problem of transaction management in a heterogeneous distributed database management system (HDBMS) has received considerable attention from the database community in recent years. The basic problem is to integrate a number of pre-existing local database management systems (DBMSs) located at different sites in a manner that allows transactions to access data residing at multiple sites and at the same time preserve the *local autonomy* of the various sites [6].

Database consistency is traditionally ensured by requiring that the concurrent execution of transactions be serializable [11]. The problem of ensuring global serializability in an HDBMS has been studied extensively in recent years. A necessary condition for maintaining global serializability is that all global transactions are serialized in the same order at all the sites at

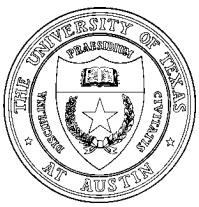
*Work partially supported by NSF grants IRI-8805215, IRI-9003341, and by grants from the IBM corporation.

MAINTAINING DATABASE CONSISTENCY IN HETEROGENEOUS DISTRIBUTED DATABASE SYSTEMS

Sharad Mehrotra
Rajeev Rastogi
Henry F. Korth
Avi Silberschatz

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188

TR-91-04 February 1991



DEPARTMENT OF COMPUTER SCIENCES
THE UNIVERSITY OF TEXAS AT AUSTIN

AUSTIN, TEXAS 78712