[5] H.F. Korth and G.D. Speegle, "Formal Model of Correctness Without Serializability", ACM-SIGMOD International Conference on Management of Data, Chicago, Illinois, June, 1988.

[6] S. Mehrotra, R. Rastogi, H. F. Korth and A. Silberschatz, "Maintaining Database Consistency in Heterogeneous Distributed Database Systems," UT Department of Computer Sciences, Technical Report, TR-91-04, 1991.

[7] S. Mehrotra, R. Rastogi, H. F. Korth and A. Silberschatz, "Proving Correctness of Non-serializable Executions," In preparation.

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

$tp_1: \quad a := 1;$ $\quad$ $\mathbf{if}(c > 0) \quad \mathbf{then} \quad b := 1$ $\qquad\qquad$ $tp_2: \quad \mathbf{if}(a > 0) \quad \mathbf{then} \quad c := b$

Let $IC = (a > 0 \rightarrow b > 0) \wedge c > 0$. The conjuncts are defined over disjoint sets of data items. Transaction programs tp_1 and tp_2 do not have fixed-structure. Consider the following schedule resulting from the execution of tp_1 and tp_2 from database state $\{(a, -1), (b, -1), (c, 1)\}$.

$$S: \quad w_1(a, 1) \quad r_2(a, 1) \quad r_2(b, -1) \quad w_2(c, -1) \quad r_1(c, -1)$$

The database state resulting from the execution of the above schedule is $\{(a, 1), (b, -1), (c, -1)\}$, which is inconsistent. Thus, PWSR schedules, resulting from the execution of transaction programs that do not have fixed-structure may not preserve integrity constraints even if conjuncts are disjoint. $\square$

4 Conclusion

We have developed a theory of non-serializable executions. Our approach exploits the knowledge of the set of data items over which the integrity constraints are defined in order to preserve database consistency in spite of sacrificing serializability. Our proof techniques were used to show that PWSR schedules, under appropriate restrictions, preserve the integrity constraints. Our results are applicable to a wide range of applications including computer-aided design and manufacturing, and heterogeneous database systems, where serializability is too strict a correctness criterion and a weaker notion of correctness based on non-serializable executions is required.

Most of the previous work on non-serializable executions resorts to informal reasoning. This is mainly due to the limitation of the transaction models being used. The transaction model we develop in this paper is suited for dealing with non-serializable executions. It differs from other standard transaction models in that operations belonging to transactions have values associated with them in addition to action and entity attributes. This new model allows us to provide a formal basis for proving that non-serializable executions preserve database consistency.

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

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

Case 1 ($p \notin t$): Trivially by IH, $read(before(t, p, S))$ is consistent.

Case 2 ($p \in t$): Let $entity(p) \in d_k$, for some $k = 1, 2, \ldots, l$. Since S is PWSR, S^{d_k} is serializable. Let a serialization order of transactions in S^{d_k} be $t_1, t_2, \ldots, t_j, t, t_{j+1}, \ldots, t_l$. As $p \in t$, $p \notin t_i$, for all $i = 1, 2, \ldots, j$. Thus, by case 1 of the induction step above, $read(before(t_i, p, S))$ is consistent, for all $i = 1, 2, \ldots, j$. Hence, since transaction programs have fixed-structure, from Lemma 4, $state(t, VS(t, p, d_k, S), S, DS)$ is consistent. By Lemma 3, $RS(before(t^{d_k}, p, S)) \subseteq VS(t, p, d_k, S)$. Thus, $read(before(t^{d_k}, p, S))$ is consistent. By IH, $read(before(t^{D-d_k}, p, S))$ is consistent. Since $d_e \cap d_f = \emptyset$, $e \neq f$, by Lemma 1, $read(before(t, p, S))$ is consistent. □

Theorem 1: Let $IC = C_1 \wedge C_2 \wedge \ldots \wedge C_l$, where IC, C_e are defined over data items in D, d_e respectively such that $d_e \cap d_f = \emptyset$, $e \neq f$. Let S be a schedule consisting of transactions resulting from the execution of transaction programs with fixed-structure. If S is a PWSR schedule, then it is strongly correct.

Proof: Let DS_1 be a consistent database state such that $legal(DS_1, S)$. Let $\{DS_1\}S\{DS_2\}$. By Lemma 5, for all $t \in \tau$, $read(t)$ is consistent (Choose p to be the last operation in the schedule). We now show that $DS_2^{d_k}$, for any $k = 1, 2, \ldots, l$ is consistent. Let $t_1, t_2, \ldots, t_n$ be a serialization order of transactions in S^{d_k}. Since $DS_1^{d_k}$ is consistent, and $d_e \cap d_f = \emptyset$, $e \neq f$, by Lemma 4, $state(t_n, d_k, DS_1, S)$ is consistent (Choose p to be the last operation in the schedule). $DS_2^{d_k}$ can be shown to be consistent by a simple application of Lemma 2. Thus, by Lemma 1, DS_2 is consistent, and hence, S is strongly correct. □

If transaction programs do not have fixed-structure or if the conjuncts are not defined over disjoint set of data items, PWSR may not preserve database consistency as is shown in Example 5 and Example 6 below.

Example 5: Consider a database containing data items $D = \{a, b, c\}$ and the following transaction programs tp_1 and tp_2.

$$tp_1: \quad b := c - 5 \qquad\qquad tp_2: \quad \begin{array}{l} temp := b; \\ a := temp + 5; \\ c := temp \end{array}$$

Let $IC = a > b \wedge a > c$. The conjuncts are not disjoint and share a data item a. Transaction programs tp_1 and tp_2 have fixed-structure. Consider the following schedule resulting from the execution of tp_1 and tp_2 from database state $\{(a, 30), (b, 10), (c, 25)\}$.

$$S: \quad r_1(c, 25) \quad r_2(b, 10) \quad w_2(a, 15) \quad w_2(c, 10) \quad w_1(b, 20)$$

The database state resulting from the execution of the above schedule is $\{(a, 15), (b, 20), (c, 10)\}$, which is inconsistent. Thus, if conjuncts are defined over sets of data items which are not disjoint, PWSR schedules may not preserve integrity constraints. □

Example 6: Consider a database containing data items $D = \{a, b, c\}$ and the following transaction programs tp_1 and tp_2.

Proof: $RS(before(t_1^d, p, S)) \subseteq d = VS(t_1, p, d, S)$. Thus, the result holds for t_1. To show that the result holds for any t_i, $i = 2, 3, \ldots, n$, we will show that if $d' \in d$, and $d' \notin VS(t_i, p, d, S)$, then $d' \notin RS(before(t_i^d, p, S))$. From the definition of the view-set of a transaction, we have the following property about data items which do not belong to a transaction's view-set. If $d' \in d$, and $d' \notin VS(t_i, p, d, S)$, then for some $j < i$, $d' \in WS(after(t_j^d, p, S))$ and for all k, $k = j+1, \ldots, i-1$, $d' \notin WS(t_k^d)$. Since S^d is serializable and t_j is serialized before t_i, if t_i reads d', then t_i must read the value of d' written by t_j. Thus, before p, t_i cannot read d'; that is, $d' \notin RS(before(t_i^d, p, S))$. □

We now use Lemma 2 and Lemma 3 to develop assertions about the database state during the execution of fixed-structure transaction programs.

Lemma 4: Let $IC = C_1 \wedge C_2 \wedge \cdots C_l$, where IC, C_e are defined over data items in D, d_e respectively such that $d_e \cap d_f = \emptyset$, $e \neq f$. Let S be a schedule resulting from the execution of transaction programs with fixed-structure such that for some k, $k = 1, 2, \ldots, l$, S^{d_k} is serializable with serialization order $t_1, t_2, \ldots, t_n$. Let p be an operation in S and DS be a database state such that $legal(DS, S)$, and DS^{d_k} is consistent. If for all $j = 1, 2, \ldots, i-1$, $read(before(t_j, p, S))$ is consistent, then $state(t_i, VS(t_i, p, d_k, S), S, DS)$ is consistent, $i = 1, 2, \ldots, n$.

Proof: The proof is by induction on the number of transactions.

Basis ($i = 1$): Trivial, as $state(t_1, d_k, S, DS) = DS^{d_k}$, which is given to be consistent.

Induction: Assume true for $i = m$, that is, if for all $j = 1, 2, \ldots, m-1$, $read(before(t_j, p, S))$, is consistent, then $state(t_m, VS(t_m, p, d_k, S), S, DS)$ is consistent. We need to show the above to be true for $i = m+1$. By IH, we know that $state(t_m, VS(t_m, p, d_k, S), S, DS)$ is consistent. By Lemma 3, $RS(before(t_m^{d_k}, p, S)) \subseteq VS(t_m, p, d_k, S)$. Since $d_e \cap d_f = \emptyset$, $e \neq f$, and $read(before(t_m, p, S))$ is given to be consistent, by Lemma 1, $state(t_m, VS(t_m, p, d_k, S), S, DS) \cup read(before(t_m, p, S))$ is consistent. As transaction program tp_m has fixed-structure, by Lemma 2, $state(t_{m+1}, VS(t_m, p, d_k, S) \cup WS(before(t_m^{d_k}, p, S), S, DS) - WS(after(t_m^{d_k}, p, S), S, DS)$ is consistent. As $VS(t_{m+1}, p, d_k, S) = VS(t_m, p, d_k, S) \cup WS(before(t_m^{d_k}, p, S)) - WS(after(t_m^{d_k}, p, S))$, $state(t_{m+1}, VS(t_{m+1}, p, d_k, S), S, DS)$ is consistent. □

Lemma 5: Let $IC = C_1 \wedge C_2 \wedge \cdots \wedge C_l$, where IC, C_e are defined over data items in D, d_e respectively such that $d_e \cap d_f = \emptyset$, $e \neq f$. Let S be a schedule consisting of transactions resulting from the execution of transaction programs with fixed-structure and p be an arbitrary operation in S. If S is a PWSR schedule, then for all transactions $t \in S$, $read(before(t, p, S))$ is consistent.

Proof: Let DS be a consistent database state such that $legal(DS, S)$. The proof is by induction on $depth(p)$.

Basis ($depth(p) = 0$): There are two cases:

Case 1 ($p \notin t$): $read(before(t, p, S)) = \emptyset$, which is consistent.

Case 2 ($p \in t$): Since $depth(p) = 0$, $read(before(t, p, S)) \subseteq DS$, which is consistent.

Induction: Assume for $depth(p) = m$, for all transactions $t \in S$, $read(before(t, p, S))$ is consistent. We need to show for $depth(p) = m+1$, for all transactions $t \in S$, $read(before(t, p, S))$ is consistent. Consider two cases.

and the lemma has been proven. □

We next associate the notion of a "state" with a transaction. The state associated with the transaction is a possible database state the transaction may have seen. The state seen by the transaction is an abstract notion and may never have been physically realized in a schedule.

Definition 4: 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 DS be a database state such that $legal(DS, S)$. The state of the database before the execution of each transaction with respect to data items in d is defined as follows.

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

$state(t_i, d, S, DS)$ 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, $read(t_i^d) \subseteq state(t_i, d, S, DS)$[2]. In Example 4, S is serializable with serialization orders t_i, t_j or t_j, t_i. With serialization order t_i, t_j,

$$state(t_j, \{a, b, c\}, S, DS) = \{(a, 0), (b, 5), (c, 5)\}.$$

However, with serialization order t_j, t_i,

$$state(t_j, \{a, b, c\}, S, DS) = \{(a, 0), (b, 10), (c, 5)\}.$$

We now introduce the notion of the *view-set* of a transaction. The view set of a transaction is defined with respect to a set of data items, and an operation in the schedule.

Definition 5: 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. The view-set of each transaction t_i, before operation p, with respect to data items in d is defined as follows.

$$VS(t_i, p, d, S) = \begin{cases} d, \text{ if } i = 1 \\ VS(t_{i-1}, p, d, S) \cup WS(before(t_{i-1}^d, p, S)) - WS(after(t_{i-1}^d, p, S)), \text{ if } i > 1 \quad \Box \end{cases}$$

The view-set of a transaction denotes the set of data items the transaction can read before an operation p in a schedule S. In the following lemma, we show that in a schedule S such that S^d is serializable, if a transaction t reads a data item $d' \in d$ before operation p, then $d' \in VS(t, p, d, S)$.

Lemma 3: 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 a schedule S. For all $i = 1, 2, \ldots, n$, $RS(before(t_i^d, p, S)) \subseteq VS(t_i, p, d, S)$.

[2]This may not be true if S^d is final-state serializable (FSR) but not view serializable [8], however it is true if S^d is view serializable, as we assume here.

- 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. $\Box$

Every serializable[1] schedule is strongly correct, but there are strongly correct schedules that are not serializable. In the next section, we show that PWSR schedules are strongly correct if transaction programs and integrity constraints are of a restricted nature. In the remainder of the paper we assume that all transaction programs and transactions are correct.

3 Predicatewise Serializability

The notion of predicatewise serializability (PWSR) was introduced as an alternative consistency criterion to serializability for applications with long-duration transactions, CAD/CAM applications, etc. [4]. In this section, we identify a set of restrictions on integrity constraints and transactions which ensure that a PWSR schedule is strongly correct. Formally, PWSR is defined as follows.

Definition 3: Let $IC = C_1 \wedge C_2 \wedge \cdots \wedge C_l$, where IC, C_e are defined over data items in D, d_e respectively. A schedule S is is said to be PWSR if for all e, $e = 1, 2, \ldots, l$, S^{d_e} is serializable. $\Box$

In order to prove that a PWSR schedule is strongly correct, we first develop conditions under which database consistency is preserved during the execution of transactions and schedules. For an arbitrary transaction, it is difficult to make any assertion about the consistency of the database state during the execution, since all we know about a transaction is that, as an atomic unit, it is correct. However, if we restrict transactions to those resulting from the execution of fixed-structured transaction programs, we can make assertions about the states which exist during its execution. In the following lemma, we state an important property of transactions resulting from execution of fixed-structured transaction programs.

Lemma 2: Let S be a schedule consisting of a transaction t which results from the execution of a fixed-structure transaction program tp (note that $S = t$). Let p be an operation belonging to schedule S and DS_1 be a database state such that $\{DS_1\}t\{DS_2\}$. If $DS_1^d \cup read(before(t,p,S))$ is consistent, then $DS_2^{d \cup WS(before(t,p,S)) - WS(after(t,p,S))}$ 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 $RS(before(t,p,S)) = RS(before(t',p',S'))$ and $WS(after(t,p,S)) = WS(after(t',p',S'))$. Since $DS_3^{RS(before(t,p,S))} = read(before(t,p,S))$ and $struct(t') = struct(t)$, $read(before(t,p,S)) = read(before(t',p',S'))$. Since writes are a function of the reads before them, t and t' result from the execution of the same transaction program tp and $struct(t') = struct(t)$, $DS_4^{d \cup WS(before(t',p',S')) - WS(after(t',p',S'))} = DS_2^{d \cup WS(before(t,p,S)) - WS(after(t,p,S))}$. Since tp is a correct transaction program, DS_4 is consistent,

[1] In this paper, unless stated otherwise, by serializability we refer to view serializability (VSR) [8].

2.3 Schedules

A schedule is 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. Let $d \subseteq D$. We denote by S^d the projection of S on data items in d.

Let seq be a subsequence of schedule S and p be an operation in S.

- We denote the subsequence of seq consisting of all the operations that precede p in S, by $before(seq, p, S)$. If p belongs to seq, then $before(seq, p, S)$ includes p.
- We denote the subsequence of seq consisting of all operations not in $before(seq, p, S)$, by $after(seq, p, S)$.
- The number of operations preceding operation p (but not including p) in schedule S is denoted by $depth(p, S)$.

Example 4: Consider the following transaction programs tp_i and tp_j.

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

Consider the schedule S below resulting from the execution of tp_i and tp_j from database state $DS = \{(a, 0), (b, 10), (c, 5), (d, 10)\}$.

$$S : \quad r_j(a, 0) \quad r_i(a, 0) \quad w_j(d, 0) \quad r_i(c, 5) \quad w_i(b, 5)$$

Note that DS is a legal database state for S, thus $legal(DS, S)$. The restriction of S to $\{a, c\}$,

$$S^{\{a,c\}} = \quad r_j(a, 0) \quad r_i(a, 0) \quad r_i(c, 5)$$

If $p = w_j(d, 0)$, then

$$\begin{aligned} before(t_j, p, S) &= \quad r_j(a, 0) \quad w_j(d, 0) \\ after(t_i, p, S) &= \quad r_i(c, 5) \quad w_i(b, 5) \\ depth(p, S) &= 2 \qquad\qquad\qquad \square \end{aligned}$$

2.4 Strong Correctness

In the traditional model, transaction programs, when executed in isolation, are assumed to be correct; that is, transactions 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 necessarily 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 2: A schedule $S = (\tau, \prec_S)$ is strongly correct iff

$WS(seq)$ 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 effects that the write operations in seq have on the database.

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

seq^d denotes the subsequence of seq consisting of all operations o such that $entity(o) \in d$.

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 2: Consider the transaction program tp_1.

$$\begin{array}{llll} tp_1 : & \textbf{if}(a = 0) & \textbf{then} & b := 0 \\ & & \textbf{else} & c := 0 \end{array}$$

The execution of tp_1 from database state $DS_1 = \{(a, 0), (b, 5), (c, 3)\}$ results in the following transaction t_1.

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

The execution of t_1 from DS_1 results in a database state $DS_2 = \{(a, 0), (b, 0), (c, 3)\}$. The following assertions can be made about transaction t_1.

$$\begin{array}{ll} RS(t_1) = \{a\} & WS(t_1) = \{b\} \\ read(t_1) = \{(a, 0)\} & write(t_1) = \{(b, 0)\} \\ legal(DS_1, t_1) & \{DS_1\}t_1\{DS_2\} \\ t_1^{\{b\}} : \quad w_1(b, 0) & struct(t_1) : \quad r_1(a) \quad w_1(b) \end{array}$$

Note that the execution of tp_1 from database state, say $DS_3 = \{(a, 10), (b, 12), (c, 15)\}$ results in a different transaction t_1'.

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

The execution of t_1' from DS_3 results in a database state, $DS_4 = \{(a, 10), (b, 12), (c, 0)\}$. □

Definition 1: 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 transactions resulting from the execution of tp from DS_1 and DS_2 respectively. □

Example 3: Consider the following transaction programs tp_1 and tp_2.

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

Transaction program tp_1 has fixed-structure, while transaction program tp_2 does not. □

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], [8]). We include, along with every operation, a value attribute, in 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.

Every transaction t has a structure associated with it denoted by $struct(t)$, which is derived from t by ignoring the values associated with the operations in t. Thus every operation o_i in $struct(t)$ is a 2-tuple $(action(o_i), entity(o_i))$. In order to discuss properties of transaction executions, we associate the following notation with a sequence seq of operations. $RS(seq)$ denotes the set of data items read by operations in seq.

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

$read(seq)$ denotes the database state "seen" as a result of the read operations in seq.

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

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 $I \models_{DS} IC$. The restriction of DS to data items in $d \subseteq D$, DS^d is consistent iff there exists a consistent database state DS_1 such that $DS_1^d = DS^d$. In the remainder of the paper, $DS \models IC$ shall denote $I \models_{DS} IC$.

Example 1: 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. $\Box$

We need to establish the relation between the consistency of database states and consistency of its subsets. This is done in the following lemma.

Lemma 1: Let $IC = C_1 \wedge C_2 \wedge \cdots \wedge C_l$, where IC, C_e are defined over data items in D, d_e respectively such that $d_e \cap d_f = \emptyset$ for all $e \neq f$. Let $d'_e \subseteq d_e$ and DS be a database state. $\bigcup_{e=1}^{l} DS^{d'_e}$ is consistent iff for all $e = 1, 2, \ldots, l$, $DS^{d'_e}$ is consistent.

Proof:

$\Leftarrow$:

If $\bigcup_{e=1}^{l} DS^{d'_e}$ is consistent, then for all $e = 1, 2, \ldots, l$, $DS^{d'_e}$ is consistent . This follows directly from the definition of database consistency.

$\Rightarrow$:

We now prove that if for all $e = 1, 2, \ldots, l$, $DS^{d'_e}$ is consistent, then $\bigcup_{e=1}^{l} DS^{d'_e}$ is consistent. Since $DS^{d'_e}$ are consistent, there exist consistent database states, DS_e, such that $DS_e^{d'_e} = DS^{d'_e}$, $e = 1, 2, \ldots, l$. Let DS_0 be a database state such that $DS_0^{d_e} = DS_e^{d_e}$, $e = 1, 2, \ldots, l$ (such a DS_0 exists since $d_e \cap d_f = \emptyset, e \neq f$). Since $DS_e \models C_e$, $DS_0^{d_e} = DS_e^{d_e}$, and C_e is defined only over data items in d_e, $DS_0 \models C_e$, $e = 1, 2, \ldots, l$. Thus, $DS_0 \models C_1 \wedge C_2 \wedge \cdots \wedge C_l$. Also, $\bigcup_{e=1}^{l} DS^{d'_e} \subseteq DS_0$. Thus, there exists a consistent database state DS_0 such that $\bigcup_{e=1}^{l} DS^{d'_e} \subseteq DS_0$. Hence, by definition of database consistency, $\bigcup_{e=1}^{l} DS^{d'_e}$ is consistent. $\Box$

Note that it is essential for the data items, over which conjuncts are defined, to be disjoint if Lemma 1 is to hold. For example, let $IC = ((a = 5 \rightarrow b = 5) \wedge (c = 5 \rightarrow b = 6))$. Consider $d'_1 = \{a\}$ and $d'_2 = \{c\}$. Let $DS^{d'_1} = \{(a,5)\}$ 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 $d_1 \cap d_2 \neq \emptyset$, database state DS_0 in the above proof does not exist.

2.2 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 knowledge of the structure of the integrity constraints and the set of data items over which they are defined in order to prove that certain non-serializable executions preserve database consistency. We also develop a transaction model suited for dealing with non-serializable executions. Using this theory we show that PWSR schedules preserve database consistency if transaction programs and integrity constraints are of a restricted nature. Our proof techniques are, however, more general and can be used to prove various other non-serializable executions preserve database consistency (see e.g., [6], [7]).

The remainder of the paper is organized as follows. In Section 2, we develop a transaction model that is suited for dealing with non-serializable executions. In Section 3, we prove that PWSR schedules preserve the database integrity constraints under certain restrictions. Concluding remarks are offered in Section 4.

2 The Transaction Model

In this section, we develop a new transaction model, that allows us to reason about non-serializable executions. We also develop a new notion of correctness as a requirement on schedules which requires more than just preservation of database consistency.

2.1 Database Consistency

In order to develop a theory of non-serializable executions, we must explicitly define what constitutes a consistent database state; which for our purpose is done in terms of *integrity constraints*.

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, denoted by DS, can be expressed as a set of ordered pairs of data items in D and their values,

$$DS = \{(d', v') : d' \in D \text{ and } v' \in Dom(d')\}.$$

DS has the property that if $(d', v'_1) \in DS$ and $(d', v'_2) \in DS$, then $v'_1 = v'_2$. The restriction of DS to data items in $d \subseteq D$, is denoted by DS^d. Thus, $DS^d = \{(d', v') : d' \in d \text{ and } (d', v') \in DS\}$. Let $d_1 \subseteq D$, $d_2 \subseteq D$, and DS_1, DS_2 be database states. The union of $DS_1^{d_1}$ and $DS_2^{d_2}$ is denoted by $DS_1^{d_1} \cup DS_2^{d_2}$. The $\cup$ operation is similar to the one traditionally defined for sets, except that $DS_1^{d_1} \cup DS_2^{d_2}$ is undefined if $(d', v'_1) \in DS_1^{d_1}$, $(d', v'_2) \in DS_2^{d_2}$, and $v'_1 \neq v'_2$.

Integrity constraints, denoted by IC, 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 [8]. 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).

On Correctness of Non-serializable Executions[*]

Sharad Mehrotra
Rajeev Rastogi
Henry F. Korth
Avi Silberschatz

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

1 Introduction

In the standard transaction model [8], 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. In the case of concurrent transaction executions, database consistency is ensured by requiring that the schedule resulting from the concurrent executions of transactions be serializable; that is, equivalent to a serial schedule [8]. Since each transaction, when executed alone, is assumed to preserve database consistency, a serializable execution preserves database consistency. This approach has the advantages of simplicity since it does not require the users to state explicitly what constitutes a consistent database state.

Serializability, however, may be too strong a correctness criterion for many applications. For example, in applications such as computer-aided design, transactions are of a long duration, implying that the serializability requirement may result in a low degree of concurrency and poor performance. Also, in a heterogeneous distributed database system environment, ensuring serializability is a difficult task due to the desire of preserving the *local autonomy* of the various sites [6].

One way to eliminate these difficulties is to relax the serializability requirement and allow non-serializable schedules that preserve database consistency. The *predicate-wise serializability (PWSR)* correctness criterion introduced in [4] is a relaxation of the serializability requirement that is applicable to environments in which transactions are of a long duration. In the nutshell, the PWSR correctness criterion states that if the database consistency constraint is expressed as a conjunction of predicates, then for each possible schedule, the projection on the set of data items in every conjunct is serializable.

In this paper, we continue our work on PWSR schedules. We first develop a theory of non-serializable executions that preserve database consistency. The cornerstone of our theory is the notion of *integrity constraints*, in terms of which database consistency is defined. We exploit the

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

ON CORRECTNESS OF NON-SERIALIZABLE EXECUTIONS

Sharad Mehrotra
Rajeev Rastogi
Henry F. Korth
Avi Silberschatz

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

TR-92-06 February 1992

DEPARTMENT OF COMPUTER SCIENCES
THE UNIVERSITY OF TEXAS AT AUSTIN

AUSTIN, TEXAS 78712