MIT/LCS/TR-210

Serializability of Concurrent Database Updates

Christos H. Papadimitriou

This blank page was inserted to preserve pagination.

MIT/LCS/TR-210

SERIALIZABILITY OF CONCURRENT DATABASE UPDATES

Christos H. Papadimitriou

March 1979

This report was prepared with the support of the National Science Foundation Grant No. MCS-77-01193 and No. MCS-77-05314. It will appear in the Journal of the ACM. A preliminary version of Sections 2 and 3 was presented at the Conference for Theoretical Computer Science at Waterloo, July 1977, in a joint paper with P.A. Bernstein and J. B. Rothnie.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LABORATORY FOR COMPUTER SCIENCE

CAMBRIDGE MASSACHUSETTS 02139

This empty page was substituted for a blank page in the original document.

The Serializability of Concurrent Database Updates*

by

Christos H. Papadimitriou
Massachusetts Institute of Technology

Abstract

A sequence of interleaved user transactions in a database system may not be *serializable*, i.e., equivalent to some sequential execution of the individual transactions. Using a simple transaction model we show that recognizing the transaction histories which are serializable is an NP-complete problem. We therefore introduce several efficiently recognizable subclasses of the class of serializable histories; most of these subclasses correspond to serializability principles existing in the literature and used in practice. We also propose two new principles which subsume all previously known ones. We give necessary and sufficient conditions for a class of histories to be the output of an efficient history scheduler; these conditions imply that there can be no efficient scheduler that outputs all of serializable histories, and also that all subclasses of serializable histories studied above have an efficient scheduler. Finally, we show how our results can be extended to far more general transaction models, to transactions with partly interpreted functions, and to distributed database systems.

*Research supported by NSF Grants MCS-77-01193 and MCS-77-05314.

This empty page was substituted for a blank page in the original document.

1. INTRODUCTION

In many situations many users may consult and update a common database. We can think of such independent user transactions as sequences of atomic database operations, interleaved with computations that are local to the user, that is, they do not affect or depend on the current state of the database. It is a function of database management to handle the update and retrieval requests made by the users in such a way so that the resulting overall process is in some appropriate sense correct. It is generally accepted--see, for example, [SLR], [SK], [EGLT], [BPR]--that the right notion of correctness in this context is that of *serializability*. A sequence of atomic user updates/retrievals is called serializable essentially if its overall effect is as though the users took turns, in some order, executing each their entire transaction indivisibly. The simplest example of a non-serializable sequence is a primitive form of a "race". Imagine two users that increment a counter by first sensing its value, and later registering an increased one. If both users retrieve the value of the counter before either of them has updated it, the resulting execution sequence--or *history*--is not serializable. This is because both possible serial executions of these transactions would have resulted in a larger total increment. Naturally, much subtler examples exist.

The appeal of serializability as a correctness criterion is quite easy to justify. Databases are supposed to be faithful models of parts of the world, and user transactions represent instantaneous changes in the world. Since such changes are totally ordered by temporal priority,

the only acceptable interleavings of atomic steps of different transactions are those that are equivalent to some sequential execution of these transactions. Another way of viewing serializability is as a tool for ensuring system correctness. If each user transaction is correct--i.e., when run by itself, it is guaranteed to map consistent states of the database to consistent states--and transactions are guaranteed to be intermingled in a serializable way, then the overall system is also correct.

In this paper we consider transactions that consist of two atomic actions: a retrieval of the values of a set of database entities--called the *read-set* of the transaction--followed by an update of the values of another set of entities--the *write-set*. This is exactly the kind of transactions handled by the system SDD-1 [BGRP], [RG]. However, the main reason for considering this model here is that it provides a nice framework for understanding and comparing very different philosophies of serializability that already exist in the literature--e.g., [BS], [SLR], [EGLT], [BGRP]. Despite its apparent simplicity, it yields a theory of serializability that is rich in combinatorial intricacies, and raises interesting complexity questions. Since our model is the most general common restriction of the models in the various references cited above, our negative results apply verbatim to those models. Furthermore, most of our positive results and characterizations are also easily generalizable to more general situations, although their proofs--in many cases their very statements--would be extremely cumbersome. Hence, we view our model as a convenient language, of the right degree of conceptual complexity, for developing and communicating our ideas about serializability, rather than a set of restrictions that enable the proofs of certain theorems.

We formalize our model of transactions in Section 2, where some preliminary results are also proved.

In Section 3 we prove that the question of whether a given sequence of read and write operations corresponding to several transactions (called a *history*) is serializable is NP-complete [AHU], [Ka]. This suggests that, most probably, there is no efficient algorithm that distinguishes between serializable and non-serializable histories.

In Section 4, we study some efficiently recognizable subsets of the set of serializable histories. In other words, we present polynomial-time "heuristics" that approximate the NP-complete predicate of serializability--in a manner quite reminiscent of efficient approximations of NP-complete optimization problems [GJ], [PS]. We show that the two-phase locking strategy [EGLT] and the protocol P3 of [BGRP] are incommensurate special cases of two more general classes called Q and DSR--the latter is related with the model of [SLR]. These two serializability principles are therefore very general (and applicable) new serialization methods. We also introduce the class SSR of histories that can be serialized without reversing the order of temporally non-overlapping transactions; it is not known whether this class is efficiently recognizable. In Section 5, we observe that the quite intricate interrelations among these interesting classes are simplified considerably if some "static" restrictions are imposed on the read- and write-sets. We point out there that the simple serializability theory of [SLR] is due to such a restriction of their model.

For all efficiently recognizable classes of histories studied in Sections 4 and 5 there is also an efficient *scheduler*; an algorithm, that is, which takes any history and transforms it to its closest (according

to some appropriate metric) history within the class considered. In Section 6 we show that this is no accident: a class of histories has an efficient scheduler if and only if it is efficiently recognizable, plus a regularity condition, namely that its set of *prefixes* is also efficiently recognizable. By this result, the complexity theory developed in Sections 3 through 5 is practically relevant, because the practical question of the existence of an efficient scheduler for a given class of histories is explicitly linked to the complexity properties of the class. Another implication is the negative result that, unless $P = NP$, there is no efficient "serializer" of histories, and hence considering efficient but more restrictive schedulers—such as the ones discussed above—is a reasonable alternative. Finally, Section 7 concludes our treatment of the subject. We discuss there a number of possible extensions of our results such as to general (multi-step) transactions and distributed databases.

2. DEFINITIONS-NOTATION

A *history* is a quadruple $h = (n,\pi,V,S)$, where n is a positive integer; π is a permutation of the set $\Sigma_n = \{R_1,W_1,R_2,W_2,\ldots,R_n,W_n\}$--that is, a one-to-one function $\pi:\Sigma_n \to \{1,2,\ldots,2n\}$--such that $\pi(R_i) < \pi(W_i)$ for $i = 1,2,\ldots,n$ (a permutation π is represented by $<\pi^{-1}(1),\pi^{-1}(2),\ldots,\pi^{-1}(2n)>$); finally, S is a function mapping Σ_n to 2^V, where V is a finite set of *variables*. Each pair (R_i,W_i) will be called a *transaction* T_i. $S(R_i)$ will be called the *read set* of T_i, and $S(W_i)$ its *write set*. We shall represent histories in a compact way by exhibiting π, with the sets $S(\cdot)$ given in brackets following each element of Σ_n. For example, the history $h = (3,<R_1,R_2,W_1,R_3,W_2,W_3>, \{x,y\},S)$ where $S(R_1) = S(R_3) = \{x\}$, $S(R_2) = \emptyset$, $S(W_3) = \{y\}$, and $S(W_1) = S(W_2) = \{x,y\}$ is represented as

$$h = R_1[x]R_2W_1[x,y]R_3[x]W_2[x,y]W_3[y].$$

The set of all histories is denoted by H.

We can think of each transaction T_i as starting with an instantaneous reading of the values in the variables in $S(R_i)$, performing a possibly lengthy local computation and then instantaneously recording the results in a different set $S(W_i)$ of variables. We do not look into the details of the exact nature of the local computation. In fact, we view each transaction T_i as a set of $|S(W_i)|$ uninterpreted $|S(R_i)|$-ary function symbols $\{f_{ij}:j = 1,\ldots,|S(W_i)|\}$. π is the sequence in which these atomic read and write operations take place. Thus, a history can be viewed as a special case of a fork-join parallel program schema, in which the local

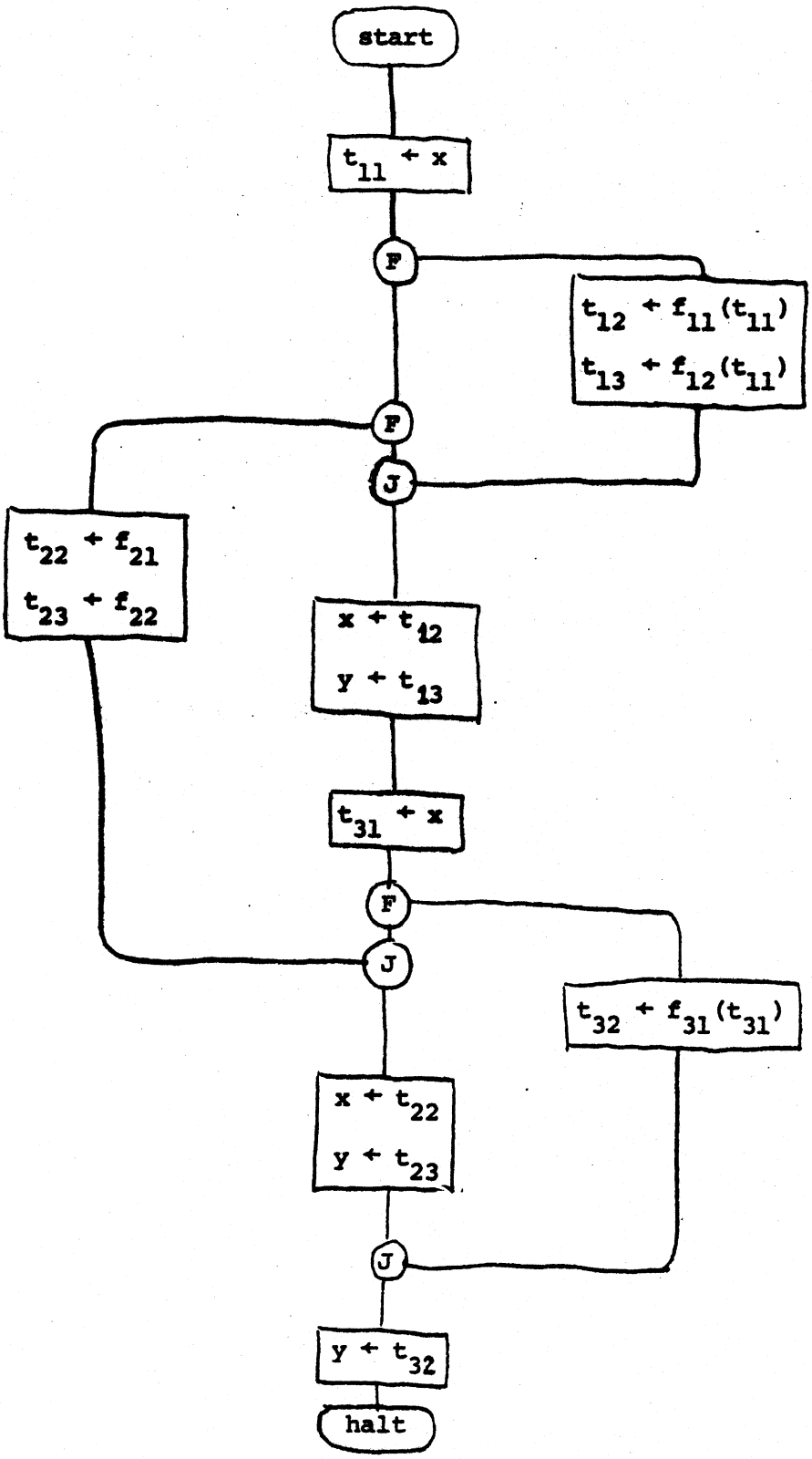


Figure 1. The history $h = R_1[x]R_2W_1[x,y]R_3[x]W_2[x,y]W_3[y]$ viewed as a parallel program schema.

computations involve a number of local temporary variables t_{ij} and are executed in parallel with other read-write operations (see Figure 1).

The *concatenation* of two histories $h_1 = (n,\pi,V,S)$, $h_2 = (m,\rho,V,T)$ is a history $h_1 \circ h_2 = (n+m,\tau,V,P)$, where $P(W_i) = S(W_i)$ if $i \leq n$, and $P(W_i) = T(W_{i-n})$ for $i > n$. Similarly, $P(R_i) = S(R_i)$ if $i \leq n$, and $P(R_i) = T(R_{i-n})$ for $i > n$. Also $\tau(W_i) = \pi(W_i)$ if $i \leq n$, and $\tau(W_i) = \rho(W_{i-n})+2n$ for $i > n$, $\tau(R_i) = \pi(R_i)$ for $i \leq n$, $\tau(R_i) = \rho(R_{i-n})+2n$ for $i > n$. In other words $h_1 \circ h_2$ is a juxtaposition of the two histories, only with the transactions of h_2 renamed. Thus, if

$$h_1 = R_1[x]R_2[y]W_2[y]R_3W_1[z]W_3[y]$$

and

$$h_2 = R_1[x,y]R_2[x]W_1[y]W_2[z] \quad ,$$

then

$$h_1 \circ h_2 = R_1[x]R_2[y]W_2[y]R_3W_1[z]W_3[y]R_4[x,y]R_5[x]W_4[y]W_5[z].$$

We say that two histories $h_1 = (n,\pi,V,S)$ and $h_2 = (n,\pi',V,S)$ are *equivalent* (written $h_1 \equiv h_2$) if and only if the corresponding schemata are (strongly) equivalent. In other words, given any set of $|V|$ domains for the variables, any set of initial values for the variables from the corresponding domains, and, furthermore, any interpretation of the functions f_{ij}, the values of the variables are identical after the execution of both histories. Notice that our definition of equivalence requires that the two histories involve the same set of transactions. Thus $h_1 = R_1[y]R_2W_2[x]W_1[x]$ is not equivalent to $h_2 = R_1[y]W_1[x]$, despite the fact that their corresponding schemata are equivalent (essentially because T_2 is "dead" in h_1). This is a matter of convenience, and little change to our derivations would be necessary in order to broaden equivalence in this sense.

To give a syntactic characterization of equivalence, it is necessary to first introduce some terminology. Let $h = (n,\pi,V,S)$ be a history. The *augmented version* of h is the history $\bar{h} = (n+2,\bar{\pi},V,\bar{S})$, where $\bar{\pi} = \langle R_{n+1},W_{n+1},\pi,R_{n+2},W_{n+2}\rangle$ and $\bar{S}(R_i) = S(R_i)$, $\bar{S}(W_i) = S(W_i)$ for $i \leq n$, and also $\bar{S}(R_{n+1}) = \bar{S}(W_{n+2}) = \emptyset$, $\bar{S}(W_{n+1}) = \bar{S}(R_{n+2}) = V$. In other words, $\bar{h}$ is h preceded by a transaction that initializes all variables without sensing any, and followed by a transaction that reads the final values of all the variables, without changing them. Suppose that $x \in S(R_i)$. We say that R_i *reads* x *from* W_j *in* h if W_j is the latest occurrence of a write symbol before R_i in $\bar{h}$ such that $x \in S(W_j)$. Notice that since $\bar{h}$ contains W_{n+1} with $S(W_{n+1}) = V$, such a write symbol always exists. The definition of a *live* transaction in h is as follows:

a. T_{n+2} is live in h.

b. If for some live transaction T_j, R_j reads a variable from W_i in h, then T_i is also live in h.

c. The only kinds of live transactions in h are defined by (a) and (b) above.

The following is now a simple syntactic characterization of history equivalence, essentially a restatement of the characterization of schema equivalence in terms of Herbrand interpretations, [LPP]:

PROPOSITION 1. Two histories $h_1 = (n,\pi,V,S)$ and $h_2 = (n,\pi',V,S)$ are equivalent if and only if they have the same sets of live transactions, and a live R_i reads x from W_j in h_1 if and only if R_i reads x from W_j in h_2. □

One of the implications of Proposition 1 is that equivalence of histories can be decided efficiently. The sets of live transactions can be found in $O(n\cdot|V|)$ time by applying the recursive definition given above, and so can the *reads from* relation for transactions. Hence we have:

COROLLARY. Equivalence of histories can be decided in $O(n\cdot|V|)$ time. □

The main theme of this paper is the notion of *serializability*. A history $h=(n,\pi,V,S)$ is *serial* if $\pi(W_i)=\pi(R_i)+1$ for all $i=1,2,\ldots,n$; in other words, a history is serial if R_i immediately preceeds W_i in it for $i=1,\ldots,n$. A history h is *serializable* (notation: $h\in SR$) if and only if there is a serial history h_s such that $h\equiv h_s$. In the next section we shall present a syntactic characterization of serializable histories analogous to (and based on) Proposition 1.

3. THE COMPLEXITY OF SERIALIZABILITY

In order to examine the complexity of the serializability problem, we need first to introduce some graph-theoretic terminology.

DEFINITION 1. A *polygraph** $P = (N,A,B)$ is a digraph (N,A) together with a set B of *bipaths*; that is, pairs of arcs--not necessarily in A--of the form $((v,u),(u,w))$ such that $(w,v) \in A$. □

Alternatively, a polygraph (N,A,B) can be viewed as a family $\mathcal{D}(N,A,B)$ of digraphs. A digraph (N,A') is in $\mathcal{D}(N,A,B)$ if and only if $A \subseteq A'$, and for each bipath $(a_1,a_2) \in B$, A' contains at least one of a_1, a_2. Polygraphs will be represented schematically as in Figure 2a. Arcs in A will be drawn as ordinary arrows, and pairs of arcs in B will be marked by a circular arc centered on their common node.

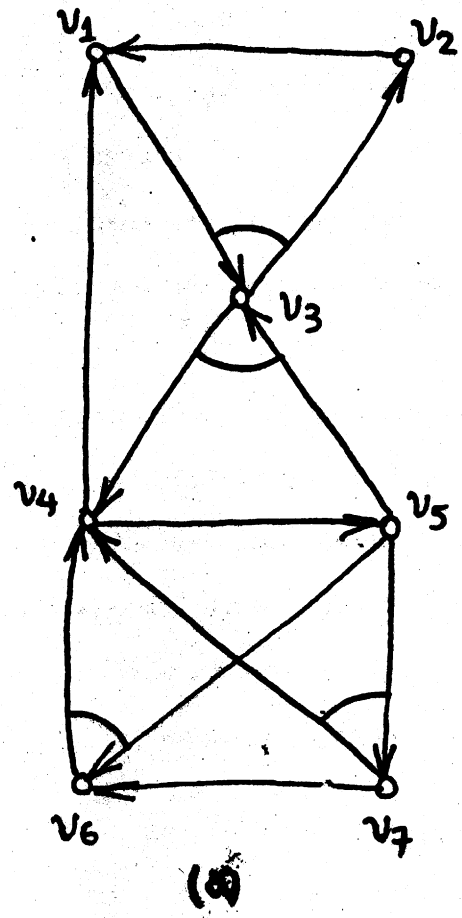
DEFINITION 2. A polygraph (N,A,B) is *acyclic* if there is an acyclic digraph in $\mathcal{D}(N,A,B)$. □

For example, the digraph of Figure 2b is both in $\mathcal{D}(N,A,B)$ and acyclic; it follows that (N,A,B) of Figure 2a is acyclic. Notice that for a polygraph (N,A,B) to be acyclic, the digraph (N,A) must definitely be acyclic.

Given any history $h = (n,\pi,V,S)$ we are going to define a polygraph $P(h) = (N,A,B)$. N is the set of live transactions of $\bar{h}$, the augmented version of h. First, A contains the arcs $\{(T_{n+1},v) : v \in N-\{T_{n+1}\}\}$, and also the

---

*We insist on this terminology only because it has already become notorious for its impropriety.



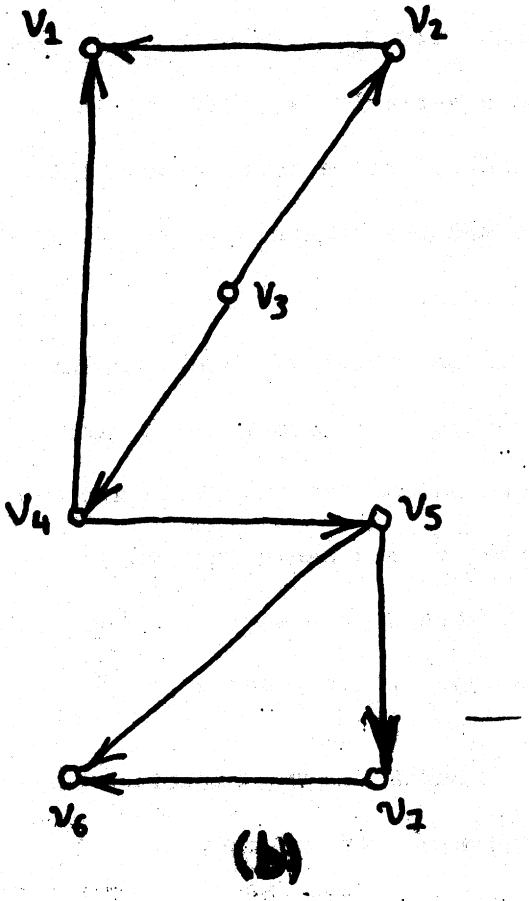


Figure 2

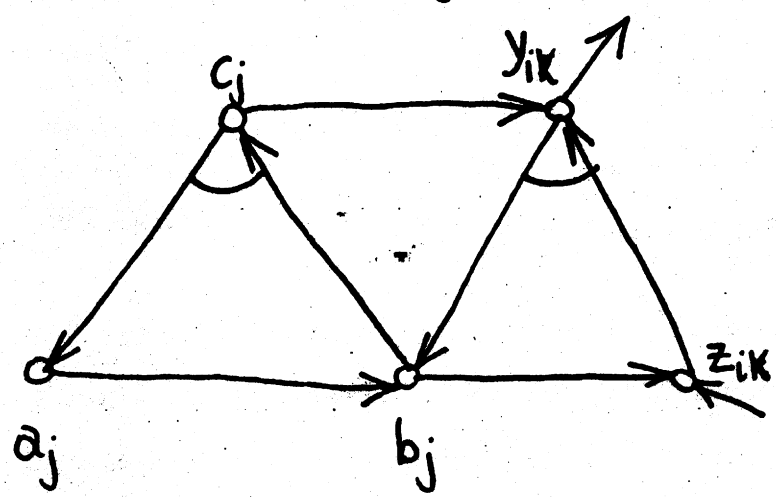


Figure 3

arcs $\{(v,T_{n+2}):v\in N-\{T_{n+2}\}\}$. Secondly, whenever transaction u reads some variable x from v in h, we add the arc (v,u) in A. Furthermore, if for a third transaction w, x is in the write-set of w, then we add the bipath ((u,w),(w,v)) in B. This concludes the construction of P(h).

Intuitively, P(h) captures a partial order that can be interpreted as "happened before", and with which any history that is equivalent to h must be consistent. Each arc (v,u) means that u read some variable from v and hence must follow it. Also, a bipath ((u,w),(w,v)) means that w writes on the same variable, and hence cannot be in between v and u; it must either precede v or follow u. This is stated as a lemma:

LEMMA 1. Two histories $h_1 = (n,\pi,V,S)$ and $h_2 = (n,\pi',V,S)$ are equivalent if and only if $P(h_1)$ and $P(h_2)$ are identical.

Proof. Both directions follow from Proposition 1 and the definition of P(h). □

LEMMA 2. A history $h = (n,\pi,V,S)$ without dead transactions is serializable if and only if P(h) is acyclic.

Proof. If h is serializable, there exists a serial history h_s such that $h \equiv h_s$ or, by Lemma 1, $P(h) = P(h_s)$. However $P(h_s) = (N,A,B)$ is acyclic. To see this, let $(T_1,\ldots,T_n)$ be ordered according to their occurrence in h_s. We construct a digraph $(N,A') \in \mathcal{D}(P(h_s))$ as follows: A' contains the arcs in A, and for each bipath $((T_i,T_j),(T_j,T_k))$ in B we add to A the arc (T_i,T_j) if $i<j$, or (T_j,T_k) if $j<k$. To

show that exactly one of these must occur, recall that in h_s T_i reads a variable $x \in S(W_j)$ from T_k, and hence $k < i$, and not $k < j < i$. Consequently the above construction yields a digraph (N,A') in $\mathcal{D}(P,A,B)$. Next, notice that (N,A') is acyclic since it is a subgraph of the total order $(T_{n+1},T_1,\ldots,T_n,T_{n+2})$. So, $P(h)$ is also acyclic.

Now, let (N,A') be an acyclic digraph in $\mathcal{D}(P(h))$. The serial history h_s resulting from topologically sorting (N,A') is then equivalent to h. This follows from Proposition 1 and from the fact that since one of the two arcs of each bipath in B is in A', all transactions in h_s read all variables from the same transaction in h as they do in h_s. □

Unfortunately, the combinatorial characterization of serial reproducibility shown in Lemma 2 does not directly suggest an efficient test. In fact, the theorem below is strong evidence that no such test exists.

THEOREM 1. Testing whether a history h is serializable is NP-complete, even if h has no dead transactions.

In order to proceed with the proof of Theorem 1 we first need another lemma. It is well known (see [AHU],[Ka]) that the satisfiability problem of Boolean formulas in conjunctive normal form with two or three literals in each clause (abbreviated SAT) is NP-complete. We can show that a more restricted version of this problem is still NP-complete. Call a clause *mixed* if it contains both variables and negations of variables, and call a formula *noncircular* if at most one of the occurrences of each variable is in a mixed clause.

LEMMA 3. SAT is NP-complete even if the formulae are restricted to be noncircular.

Proof. Consider any instance F of SAT and a variable x in it. Let m be the number of occurrences of x in the formula F, and let $x_1, x_2, \ldots, x_m$ be new variables. We replace x in its first occurrence by x_1, in its second by $\bar{x}_2$, in its third by x_3, etc. Finally, we add the clauses $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3) \wedge \ldots$, which is the conjunctive normal form of $x_1 \equiv \bar{x}_2 \equiv x_3 \equiv \bar{x}_4 \equiv \cdots$. Repeating this for all variables, we observe that the resulting formula is trivially noncircular, and the construction requires only a polynomial amount of time. □

Proof of Theorem 1. The set of SR histories is definitely in NP, since to show that h is SR, one only needs to construct a serial history h_S (of length not greater than that of h), and check by Proposition 1 that h and h_S are equivalent.

We will next show that a known NP-complete problem, the noncircular SAT problem of Lemma 3 above, reduces to SR-testing in polynomial time.

Given any such formula F, we are going to construct a polygraph $P_F = (N,A,B)$ such that P_F is acyclic if and only if F is satisfiable. We will then show that P_F can be considered as $P(h)$ for a suitable history h, without dead transactions. In view of Lemma 2, this will conclude the proof.

We start from the construction of $P_F = (N,A,B)$. F has m clauses $C_1, \ldots, C_m$ and involves n Boolean variables $x_1, \ldots, x_n$. Each clause C_i consists of three literals $\lambda_{i1} \vee \lambda_{i2} \vee \lambda_{i3}$, where λ_{jk} is either a variable or a negation of one. N contains the nodes a_j, b_j, c_j for each variable

x_j, and y_{ik}, z_{ik}, $k = 1,..,m_i$ for each clause C_i with m_i literals. For each variable x_j we add the arc (a_j,b_j) to A, and the bipath $((b_j,c_j),(c_j,a_j))$ to B. For each clause C_i, we add the arcs $(y_{ik},z_{i,k+1})$ (addition mod m_i) to A. Finally, if $\lambda_{ik} = x_j$, we add the arcs (c_j,y_{ik}) and (b_j,z_{ik}) to A, and the bipath $((z_{ik},y_{ik}),(y_{ik},b_j))$ to B. If $\lambda_{ik} = \bar{x}_j$, then we add the arcs (z_{ik},c_j) and (y_{ik},a_j) to A, and the bipath $((a_j,z_{ik}),(z_{ik},y_{ik}))$ to B. For example, if the literal λ_{ik} is x_j, the subpolygraph of Figure 3 will appear in P_F.

Finally, we add to N the nodes n_0,n_c and n_f, together with the arcs $(n_0,n),(n,n_c)$ and (n,n_f) for all $n \in N-\{n_0,n_c,n_f\}$, and also the arc (n_c,n_f). This concludes the construction of P_F. In Figure 4a we illustrate the construction for the Boolean formula